UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Berechnungskomplexitäten von Varianten des Subset-Sum-Problems

Computational Complexities of Variants of the Subset Sum Problem

Bachelorarbeit

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Max Bannach

ausgegeben und betreut von
Prof. Dr. Till Tantau

mit Unterstützung von
Dipl.-Inf. Christoph Stockhusen

Lübeck, den 23. Oktober 2012

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig und ausschließlich unter Verwendung der angegebenen Quellen angefertigt habe.

Ort, Datum Unterschrift

Zusammenfassung

In dieser Arbeit betrachten wir das Problem SUBSETSUM, bei dem aus einer gegebenen Menge von natürlichen Zahlen eine Teilmenge bestimmt werden soll, sodass die Elemente dieser Teilmenge in der Summe eine vorgegebene Zahl ergeben. Die Komplexität von SUBSETSUM ist für die Informatik interessant, denn aufgrund des generischen Aufbaus von SUBSETSUM lassen sich viele weitere Probleme darauf zurückführen. In dieser Arbeit soll die Komplexität von mehreren Varianten von SUBSETSUM untersucht werden, denn obwohl fast alle Varianten des Problems in der üblichen binären Codierung NP-vollständig sind, haben diese Probleme oft unterschiedliche Komplexitäten, wenn wir unäre Kodierungen betrachten. In dieser Arbeit werden einige natürliche Varianten dieses Problems diesbezüglich untersucht und entsprechende Resultate präsentiert.

Abstract

In this thesis we take a look at the problem SUBSETSUM, in which a subset out of a set of natural numbers must be chosen in a way, that the sum of these elements equals a given number. A lot of natural problems can be reduced to SUBSETSUM because of the simple structure of the problem, therefore the complexity of SUBSETSUM is interesting for computer science. In this thesis we will analyze a few versions of SUBSETSUM because all these versions are NP complete if we consider a default binary encoding, but a few of them lie in different complexity classes if we consider them with a unary encoding. In this thesis we will study a few natural versions of the problem and examine them in this regard and present the results.

Inhaltsverzeichnis

1 Einführung

Eine zentrale Aufgabe der Theoretischen Informatik ist das Beurteilen der Schwierigkeit von Problemen. Dabei ist – zumindest in der Informatik – der Schwierigkeitsgrad eines Problems der Rechenaufwand, den ein Computer beim Lösen des Problems hat. Dieser Rechenaufwand spiegelt sich im Verbrauch von verschiedenen Ressourcen wider. Besonders wichtige Ressourcen sind Zeit und Platz, denn für große Eingaben liefern Algorithmen mit hohem Zeitbedarf erst nach mehreren Jahren Ergebnisse und der Speicher moderner Computer ist bei unachtsamen Platzverbrauch schnell ausgelastet [7, 12]. Es ist ein Trugschluss anzunehmen, dass diese Ressourcen an Wert verlieren, wenn unsere Computer schneller werden und über mehr Speicher verfügen, denn auch die von uns betrachteten Probleminstanzen werden immer größer – und zwar so drastisch, dass der Ressourcenverbrauch sogar noch kritischer betrachtet werden muss. Um den Sachverhalt des Ressourcenverbrauches formal zu beschreiben, sprechen wir von sogenannten *Komplexitätsklassen*. Eine solche Komplexitätsklasse ist eine Familie von Problemen, die alle von Maschinenmodellen, die der selben *Ressourcenschranke* unterliegen, entschieden werden können. Solch eine Ressourcenschranke legt im Bezug zur Eingabe beispielsweise fest, wie viel Zeit oder Platz eine Turingmaschine zum Lösen eines Problems maximal verwenden darf. So ist zum Beispiel die Komplexitätsklasse P die Klasse der Probleme, die von einer Turingmaschine, die einer im Bezug zur Eingabelänge polynomiellen Zeitschranke unterliegt, gelöst werden können. Viele Probleme, die in binärer Codierung in der Komplexitätsklasse P oder einer höheren Klasse liegen, fallen in einer unären Codierung in eine kleinere Komplexitätsklasse; weil des Weiteren L und NL die wichtigen Unterklassen von P sind, befassen wir uns in dieser Arbeit mit Berechnungskomplexitäten, die sich auf den Verbrauch von Platz beziehen. Eine wichtige Platzklasse ist zum Beispiel L, die Menge der Probleme, die von einer Turingmaschine mit im Bezug zur Eingabelänge logarithmischen Platzschranke entschieden werden können.

In dieser Arbeit wollen wir uns mit Varianten des Problems SUBSETSUM beschäftigen. Bei diesem Problem sollen aus einer Menge von gegebenen na-

türlichen Zahlen Elemente so ausgewählt werden, dass diese in der Summe einen vorgegebenen Zielwert bilden. Formal lässt sich SUBSETSUM wie folgt beschreiben:

► Problem 1 (Subset-Sum)
Eingabe: Ein Vektor $\vec{a} \in \mathbb{N}^p$ mit $p \in \mathbb{N}$ sowie eine Zielsumme $b \in \mathbb{N}$.
Gesucht: Ein $\vec{x} \in \{0,1\}^p$, sodass die folgende Gleichung gilt:

$$\sum_{i=1}^{p} a_i \cdot x_i = b.$$

Sprache: $\text{SUBSETSUM} = \{\text{bin}(a_1)\#\text{bin}(a_2)\#\cdots\#\text{bin}(a_p)\#\#\text{bin}(b) \mid \exists I \subseteq \{1,\ldots,p\}.\ \sum_{i\in I} a_i = b\},$

$\text{UNARY-SUBSETSUM} = \{0^{a_1}10^{a_2}1\ldots10^{a_p}10^b \mid \exists I \subseteq \{1,\ldots,p\}.\ \sum_{i\in I} a_i = b\}.$ ◁

Bei SUBSETSUM handelt es sich um ein Problem, das sowohl in der Komplexitätstheorie als auch in der Kryptologie sehr bekannt ist. Es besitzt einen vergleichweise simplen und intuitiven Aufbau, weshalb es sich in der Komplexitätstheorie als Grundlage für viele Untersuchungen anbietet. In der Kryptologie nutzt man beispielsweise die NP-Vollständigkeit des Problems für verschiedene Verfahren aus [11]. Wir beschäftigen uns in dieser Arbeit mit dem Platzverbrauch von SUBSETSUM und seinen Varianten, denn Erkenntnisse aus der Theoretischen Informatik bringen ein Phänomen zum Vorschein: so sind alle Varianten von SUBSETSUM in der üblichen binären Codierung NP-vollständig, doch ändern wir die Codierung auf ein unäres System, so lassen sich die Varianten in unterschiedliche Platzklassen einordnen. Das Problem UNARY-SUBSETSUM lässt sich beispielsweise in die Komplexitätsklasse TC^0 ordnen. Der Beweis basiert auf den Ergebnissen von Elberfeld, Jakoby und Tantau [5]. Auch wenn der Beweis an dieser Stelle noch nicht klar ist, so können wir uns intuitiv in etwa vorstellen, dass die Berechnung in einer kleinen Platzklasse dadurch möglich wird, dass wir die Eingabe durch eine unäre Darstellung drastisch vergrößern.

Um so erstaunlicher ist es, dass eben dieser Effekt beim verwandten Problem PARTIAL-UNARY-SUBSETSUM nicht auftritt. Bei diesem Problem sollen wie bei SUBSETSUM Elemente aus einer Menge von natürlichen Zahlen ausgewählt werden, sodass diese in der Summe einen vorgegebenen Zielwert bilden. Zusätzlich soll die Summe dieser Elemente einer gewisse unteren Schranke genügen, während die Elemente ausgewählt werden. Das Problem PARTIAL-UNARY-SUBSETSUM ist auf der nächsten Seite formal beschrieben.

▶ Problem 2 (Partial-Subset-Sum)
Eingabe: Zwei Vektoren $\vec{a}, \vec{c} \in \mathbb{N}^p$ mit $p \in \mathbb{N}$ sowie eine Zielsumme $b \in \mathbb{N}$.
Gesucht: Ein $\vec{x} \in \{0,1\}^p$, sodass die folgenden Gleichungen gelten:

$$\sum_{i=1}^{p} a_i \cdot x_i = b,$$
$$\sum_{i=1}^{\nu} a_i \cdot x_i \geqslant c_\nu \quad \text{für alle } \nu \in \{1,2,\ldots,p\}.$$

Sprache: PARTIAL-SUBSETSUM
$= \{\, \mathrm{bin}(a_1)\#\mathrm{bin}(c_1)\#\cdots\#\mathrm{bin}(a_p)\#\mathrm{bin}(c_p)\#\#\mathrm{bin}(b) \mid$
$\exists I \subseteq \{1,\ldots,p\} \,.\, \sum_{i\in I} a_i = b \wedge \forall_{\nu \leqslant p} \,.\, \sum_{i\in I, i\leqslant \nu} a_i \geqslant c_\nu \,\}.$
PARTIAL-UNARY-SUBSETSUM
$= \{\, 0^{a_1}10^{c_1}110^{a_2}10^{c_2}11\ldots110^{a_p}10^{a_p}1110^{b} \mid$
$\exists I \subseteq \{1,\ldots,p\} \,.\, \sum_{i\in I} a_i = b \wedge \forall_{\nu \leqslant p} \,.\, \sum_{i\in I, i\leqslant \nu} a_i \geqslant c_\nu \,\}.$ ◁

Bereits 1979 konnte Monien zeigen, dass PARTIAL-UNARY-SUBSETSUM ein NL-vollständiges Problem ist [9]. Der Beweis von Monien ist sehr technisch und daher schwer zu erfassen, wir wollen ihn in dieser Arbeit genau analysieren und verständlich notieren.

Gilt, wie allgemein angenommen, $TC^0 \neq L$ und $L \neq NL$, so liegen verschiedene Varianten von UNARY-SUBSETSUM in unterschiedlichen Komplexitätsklassen. Wir tragen diese Erkenntnisse detailliert zusammen und untersuchen die Zusammenhänge weiter; dazu stellen wir verschiedene Varianten von UNARY-SUBSETSUM vor. Für ein gutes Verständnis dieser Untersuchungen folgt in Abschnitt 2 zunächst eine Einführung in Logik, Graphentheorie und Komplexitätstheorie; anschließend werden wir dort auch die zentralen Sätze von Elberfeld, Jakoby und Tantau vorstellen, mit denen wir viele Varianten von UNARY-SUBSETSUM untersuchen können. Anschließend betrachten wir in Abschnitt 3.1 unter Verwendung der Sätze von Elberfeld, Jakoby und Tantau die Komplexität von UNARY-SUBSETSUM. In Abschnitt 3.2 folgt dann der für ein besseres Verständnis überarbeite Satz von Monien, der die NL-Vollständigkeit von PARTIAL-UNARY-SUBSETSUM zeigt. Im folgenden Abschnitt untersuchen wir verschiedene mehrdimensionale Varianten von UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM, wie das bekannte Problem UNARY-KNAPSACK. Alle in diesem Abschnitt betrachten Varianten von UNARY-SUBSETSUM werden wir als TC^0-vollständig klassifizieren, während alle Varianten von PARTIAL-UNARY-SUBSETSUM sogar NL-vollständig sind.

In Abschnitt 4.2 untersuchen wir Varianten von PARTIAL-UNARY-SUBSETSUM mit veränderten Beschränkung, wie zum Beispiel das durch eine zusätzliche obere Schranke erweiterte Problem TUBE-UNARY-SUBSETSUM. Diese härtere Beschränkung ändert allerdings nichts an der Komplexität und auch

TUBE-UNARY-SUBSETSUM werden wir als NL-vollständig klassifizieren. Eine weitere Variante die wir in diesem Abschnitt betrachten ist das Problem WAYPOINT-UNARY-SUBSETSUM, in welcher wir die untere Schranke durch mehre Zwischensummen – die alle zu erreichen sind – ersetzen. Diese Variante von PARTIAL-UNARY-SUBSETSUM werden wir als TC^0-vollständig klassifizieren.

Abschließend betrachten wir in Abschnitt 4.3 Varianten von den Probleme UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM über anderen Körpern. Hier ersetzen wir die natürlichen Zahlen der Eingabe durch ganze bzw. rationale Zahlen. Wir werden zeigen, dass die Varianten über den ganzen Zahlen in dieselben Komplexitätsklassen fallen, wie die Ausgangsprobleme. Die Komplexität von einer über den rationalen Zahlen definierten Variante von UNARY-SUBSETSUM konnte in dieser Arbeit jedoch nicht vollständig bestimmt werden, denn diese Änderung macht das Problem deutlich komplexer. Wir werden diese Variante von UNARY-SUBSETSUM als NL-hart klassifizieren. Eine Übersicht aller in dieser Arbeit betrachteten Varianten und deren Komplexitäten findet sich in Abschnitt 5.

2 Grundlagen und Notationen der Logik sowie der Graphen- und Komplexitätstheorie

In dieser Arbeit nutzen wir viele Begrifflichkeiten aus Logik, Graphentheorie und Komplexitätstheorie, die wir in den folgenden drei Abschnitten zunächst festlegen. Anschließend stellen wir im letzten Abschnitt die Sätze von Bodlaender und Courcelle in der Version von Elberfeld, Jakoby und Tantau vor, mit welchen wir im weiteren Verlauf der Arbeit die Komplexität von verschiedenen Varianten von UNARY-SUBSETSUM bestimmen können.

2.1 Monadische Logik zweiter Stufe

Werkzeuge der Logik sind von großem Interesse für uns, denn wir können die Eingaben der von uns betrachteten Probleme als logische Strukturen beschreiben und anschließend Aussagen über die Komplexität der Probleme treffen, indem wir den Platzverbrauch einer Turingmaschine bestimmen, die logische Formeln über diesen Strukturen auswertet. Wie wir Logik und Komplexität genau verbinden, wird in Abschnitt 2.4 beschrieben. In diesem Abschnitt führen wir zunächst die logische Struktur ein und definieren, wie wir Aussagen für diese formulieren können. Wir orientieren uns an den Notationen aus den entsprechenden Kapiteln aus dem Buch von Flum und Grohe [6], eine gute Ausarbeitungen im Bezug zu der hier betrachteten Thematik findet sich bei Stockhusen [15].

Eine *Signatur* τ ist ein Tupel von *Relationssymbolen* mit einer Stelligkeit größer gleich 1 und *Konstantensymbolen*. Wir bezeichnen die Relationssymbole mit lateinischen Großbuchstaben und die Konstantensymbole mit lateinischen Kleinbuchstaben. Um zu beschreiben, dass ein Relationssymbol R die Stelligkeit i besitzt und in τ liegt, notieren wir verkürzend $R^i \in \tau$. Entsprechend schreiben wir $r^0 \in \tau$ für ein Konstantensymbol r^0 in τ.

Eine Signatur τ besitzt noch keine Bedeutung, erst eine *Struktur* S über τ definiert das *Universum* U und weist den Symbolen aus τ eine Bedeutung zu. Das Universum ist eine nichtleere und in dieser Arbeit immer endliche

Menge der Elemente, die in der logischen Struktur betrachtet werden. Beispielsweise könnte $U \subseteq \mathbb{N}$ gelten, aber auch jede andere Art von Objekten ist möglich. In S wird jedem Relationssymbol R mit Stelligkeit i aus τ eine *Relation* $R^S \subseteq U^i$ zugewiesen, weiter wird jedem Konstantensymbol $r^0 \in \tau$ ein Element r^S des Universums zugeordnet.

▶ Beispiel 3 (Binärstrings)
Wir können einen Binärstring $a = a_1 a_2 \dots a_p$ als logische Struktur B über einer Signatur $\tau_B = (U, \leqslant^2, I_1^1)$ beschreiben. Dabei bezeichnen wir in der Struktur B mit dem Universum $U = \{1, 2, \dots, p\}$ die natürlichen Zahlen von 1 bis p. Dem Relationsymbol $\leqslant^2$ ordnen wir die intuitive Bedeutung über den natürlichen Zahlen zu. Die Relation I_1^B beschreibt die Positionen, an denen der Binärstring a eine 1 besitzt.

Wir können zum Beispiel den Binärstring 011001 durch die logische Struktur $S = (U, \leqslant^B, I_1^B)$ mit $U = \{1, 2, 3, 4, 5, 6\}$ und $I_1^B = \{2, 3, 6\}$ beschreiben. ◁

Mit logischen Signaturen und Strukturen können wir Probleme und andere Sachverhalte gezielt beschreiben, jedoch können wir noch keine Aussagen über diese formulieren und ihre Gültigkeit überprüfen. Insbesondere für das Lösen von über Logik beschriebenen Problemen ist dies aber von fundamentaler Bedeutung. Um daher solche Aussagen formulieren zu können, nutzen wir *logische Formeln*.

Wir bezeichnen *Variablensymbole erster Stufe* (*Elementarvariablen*) mit lateinischen Kleinbuchstaben, die in der betrachteten Signatur nicht vorkommen. Über einer Signatur τ mit $R^n \in \tau$ und Elementarvariablen oder Konstantensymbolen $x_1, \dots, x_n, x, y$ sind dann $(x = y)$, $(x \neq y)$ und $R(x_1, \dots, x_n)$ *atomare logische Formeln*. Die *logischen Formeln erster Stufe* (*FO-Formeln*) sind nun induktiv definiert. Seien ϕ und ψ zwei logische Formeln erster Stufe, dann sind auch Verknüpfungen durch logische Operatoren $(\neg\phi)$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \to \psi)$, $(\phi \leftrightarrow \psi)$ und Verknüpfungen durch Quantoren $(\exists x(\phi))$, $(\forall x(\phi))$ FO-Formeln.

Sind $x_1, \dots, x_n$ Elementarvariablen oder Konstantensymbole, so ist R ein *Variablensymbol zweiter Stufe* (eine *Relationsvariable*), wenn R nicht in der Signatur vorkommt und R eine n-Stellige Relation ist. Wir notieren solche Relationsvariablen mit lateinischen Großbuchstaben und nennen $R(x_1, \dots, x_n)$ eine *logische Formel zweiter Stufe* (*SO-Formeln*). Die Menge der SO-Formeln definiert sich dann induktiv wie die Menge der FO-Formeln mit folgender Erweiterung: ist R eine Relationsvariable und ϕ eine logische Formel zweiter Stufe, so sind auch $(\exists R(\phi))$ und $(\forall R(\phi))$ SO-Formeln. Relationsvariablen der Stelligkeit 1 heißen *monadische Relationsvariablen*, entsprechend sind SO-Formeln, deren Relationsvariablen alle monadisch sind, *monadische logische Formeln zweiter Stufe* (*MSO-Formeln*).

In einer logischen Formel ϕ heißen alle Variablen x, die in einer Formel der Form $(\exists x(\varphi))$ oder $(\forall x(\varphi))$ vorkommen, *gebunden*. Sind $x_1, \ldots, x_n$ die *freien* Variablen einer logischen Formel ϕ, so schreiben wir auch $\phi(x_1, \ldots, x_n)$. Die *Belegung* einer logischen Struktur S ist eine Funktion α, die jeder freien Elementarvariable ein Element und jeder freien n-stelligen Relationsvariable eine n-stelligen Relation über dem Universum zuordnet. Erfüllt eine Struktur S mit einer Belegung α eine logische Formel ϕ, so ist S für α ein *Modell* von φ, wir schreiben $(S, \alpha) \models \phi$. Ist S für alle Belegungen ein Modell von ϕ, so schreiben wir kurz $S \models \phi$.

Ein für uns wichtiges Werkzeug ist die FO(BIT)-Reduktion, eine Reduktion in Logik erster Stufe unter Zunahme der BIT-Relation. Die Relation $\text{BIT}(x, y)$ wird zu wahr ausgewertet, wenn die Binärdarstellung von x an Position y eine 1 besitzt. Immerman zeigte zum einen, dass die Komplexitätsklasse AC^0 (siehe Abschnitt 2.3) der Menge der Probleme entspricht, die sich in FO-Logik formulieren lassen, und dass diese Probleme eine stabilere Komplexitätsklasse bilden, wenn die BIT-Relation hinzugenommen wird. Er zeigte außerdem, dass die Hinzunahme der BIT-Relation äquivalent zu der Hinzunahme der ADD- und TIMES-Relation ist [8]. Die Relationen ADD und TIMES sind dabei wie folgt definiert:

- $\text{ADD}(x, y, z)$ wird zu wahr ausgewertet, wenn $x + y = z$ gilt;
- $\text{TIMES}(x, y, z)$ wird zu wahr ausgewertet, wenn $x \cdot y = z$ gilt.

Damit wir logische Formeln leichter lesen können, führen wir die *Bindungsstärke* für Operatoren und Quantoren ein. Diese ist in absteigender Reihenfolge $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \exists, \forall$. Somit ist zum Beispiel $\varphi \wedge \phi \rightarrow \psi$ äquivalent zu $(\varphi \wedge \phi) \rightarrow \psi$. Weiter nutzen wir die *Punktnotation*, wobei wir einen Punkt anstelle einer öffnenden Klammer setzen. Diese Klammer wird am Ende der längsten formal korrekten Formel geschlossen, aber nicht notiert. Wir können somit $\varphi \rightarrow . \phi \wedge \psi$ anstelle von $\varphi \rightarrow (\psi \wedge \phi)$ schreiben.

Elberfeld, Jakoby und Tantau führten das *Lösungshistogramm* $\text{histogram}(S, \varphi)$ für eine logische Struktur S und eine MSO-Formel $\varphi(X_1, \ldots, X_n)$ mit freien Relationsvariablen $X_1, \ldots, X_n$ ein [4]. Dieses ist ein n-dimensionale Array aus natürlichen Zahlen, dessen $(i_1, \ldots, i_n)$-ter Eintrag angibt, wie viele erfüllende Belegungen es im Universum gibt, wenn die Kardinalität der Belegungen der Relationsvariablen durch $|X_1| = i_1$, $|X_2| = i_2$, …, $|X_n| = i_n$ gegeben ist. Wir geben den $(i_1, \ldots, i_n)$-ten Eintrag mit $\text{histogram}(S, \varphi)[i_1] \ldots [i_n]$ an. Ein solches Lösungshistogramm beinhaltet viele Informationen über S und φ, insbesondere können wir ablesen, wie viele erfüllende Belegungen es insgesamt gibt, indem wir einfach alle Einträge von $\text{histogram}(S, \varphi)$ aufsummieren.

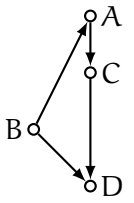
2.2 Graphen und Baumzerlegungen

Graphen sind ein mächtiges mathematisches Werkzeug zum Modellieren von Problemen, Sachverhalten und Prozessen. Gerade weil Graphen so vielseitig sind, hat sich die Theorie um sie zu einem wichtigen Teilgebiet der Mathematik und Informatik entwickelt. Für diese Arbeit ist das Modellieren von Eingaben für die von uns betrachteten Probleme von hohem Wert, denn gelten für die so entstehenden Graphenstrukturen gewisse Eigenschaften, so können wir diese mit Mitteln aus der Logik in bekannter Komplexität auswerten und so Rückschlüsse auf die Komplexität der modellierten Probleme ziehen. Doch bevor wir in Abschnitt 2.4 solche Komplexitätsaussagen formulieren, wollen wir in diesem Abschnitt zunächst den Begriff des Graphen und unsere Notationen einführen. Wir orientieren uns am Buch von Diestel [3], in dem der interessierte Leser einen detaillierteren Einstieg in die vielen Teilgebiete der Graphentheorie geboten bekommt.

Ein *gerichteter Graph* ist eine logische Struktur $G = (V, E^G)$ über der logischen Signatur $\tau_G = (E^2)$. Wir nennen $V = \{1, \ldots, n\}$ die Menge der *Knoten* und $E^G = \{(x, y) \mid x, y \in V\}$ die *Kantenrelation*. Verkürzend schreiben wir $V(G)$ und $E(G)$, um die Menge der Knoten bzw. Kanten eines Graphen G anzugeben. Sind zwei Knoten durch eine Kante verbunden, so nennen wir die Knoten *adjazent* zueinander. Sind zwei Kanten durch einen Knoten verbunden, so nennen wir auch diese Kanten *adjazent*. Geht eine Kante von einem Knoten aus, oder mündet in diesen, dann sagen wir: die Kante ist *inzident* zu diesem Knoten. In einer Kante $(x, y) \in E$ nennen wir x den *Startknoten* und y den *Endknoten*. Die Anzahl der zu einem Knoten x inzidenten Kanten nennen wir *Grad* $d(x)$ von x. Die Anzahl der Kanten, die x als Startknoten besitzen, ist der *Ausgangsgrad* $d_{out}(x)$ von x, und entsprechend ist die Anzahl der Kanten, die x als Endknoten besitzen, der *Eingangsgrad* $d_{in(x)}$ von x. Existiert eine Folge von Kanten und Knoten, sodass man dieser folgend von einem Knoten x zu einem Knoten y gelangt, so existiert ein *Pfad* von x nach y. Ein Graph heißt *zusammenhängend*, wenn zwischen je zwei Knoten x und y ein Pfad von x nach y oder ein Pfad von y nach x existiert. Ist ein Graph nicht zusammenhängend, so zerfällt er in *Komponenten*, diese Komponenten sind zusammenhängende Graphen. Für einen Graphen $G = (V, E^G)$ ist $G - x$ für einen Knoten $x \in V$ definiert durch das Entfernen von x aus V und dem Löschen aller zu x inzidenten Kanten aus E. Die Minusoperation einer Kante löscht nur diese Kante aus dem Graphen.

Eine praktische Eigenschaft von Graphen ist, dass sie sich sehr gut visuell darstellen lassen, indem die Knoten als Punkte und die Kanten als Pfeile zwischen Knoten gezeichnet werden.

► Beispiel 4 (Abhängigkeiten zwischen Prozessen)
Ein typisches Szenario für den Einsatz von Graphen ist die Modellierung von Abhängigkeiten zwischen Prozessen. Seien A, B, C und D vier Prozesse, zwischen denen Abhängigkeiten der folgenden Form bestehen: A muss vor C, B vor A und D, und C vor D ausgeführt werden. Wir können diesen Sachverhalt als Graph modellieren, indem wir $V = \{A, B, C, D\}$ und $E = \{(A, C), (B, A), (B, D), (C, D)\}$ setzen. Der beschriebene Graph ist in der Abbildung rechts zu sehen. Um nun zum Beispiel eine gültige Reihenfolge der Prozesse zu ermitteln, muss nur ein Pfad im konstruierten Graphen ermittelt werden, der alle Knoten besucht. In diesem Beispiel ist die Lösung $B \to A \to C \to D$ aus der Grafik ersichtlich. ◁



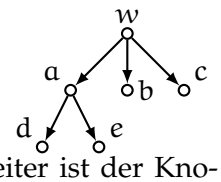
Ein Spezialfall von gerichteten Graphen sind jene mit einer *symmetrischen Kantenrelation*, wir nennen diese Graphen *ungerichtet*. Zeichnen wir einen ungerichteten Graphen, so zeichnen wir statt der zwei Pfeile nur eine Linie zwischen zwei adjazenten Knoten.

In vielen Szenarien ist es wünschenswert, mehr Informationen in einem Graph zu modellieren, um unterschiedliche Teilbeziehungen der Knoten und Kanten stärker zu verdeutlichen. Wichtige Werkezuge sind dabei die *Knotenfärbung* und die *Kantenfärbung*. Sprechen wir von Graphen mit einer Knotenfärbung, so sprechen wir von logischen Strukturen G über der Signatur $\tau_G = (E^2, C_1^1, \ldots, C_n^1)$. Die Struktur $G = (V, E^G, C_1^G, \ldots, C_n^G)$ weist den Relationen C_1^G bis C_n^G eine Farbe zu, sodass $C_i^G(x)$ für einen Knoten $x \in V$ gilt, wenn x mit der Farbe C_i gefärbt ist. Ein Graph mit gefärbten Kanten ist entsprechend eine Struktur über der Signatur $\tau_g = (E^2, C_1^2, \ldots, C_n^2)$, wobei die Relation $C_i^G(x, y)$ für eine Kante $(x, y) \in E^G$ gilt, wenn (x, y) mit der Farbe C_i gefärbt ist. Graphen können gleichzeitig eine Knoten- und eine Kantenfärbung besitzen. Indem wir ein hinreichend großes n wählen, können wir die Färbung auch als Beschriftung auffassen, um verschiedene Informationen in den Graphen zu kodieren.

Enthält ein ungerichteter Graph keinen Kreis, so nennen wir ihn *Wald*, ist ein Wald zusammenhängend, so ist er ein *Baum*. Ein gerichteter Graph G ist ein Wald oder Baum, wenn der Graph – der entsteht, wenn wir die Kantenrelation von G auf eine symmetrische Kantenrelation erweitern – ein Wald oder Baum ist. Ein gerichteter Baum heißt *Wurzelbaum*, wenn genau ein Knoten w mit $d_{in}(w) = 0$ oder $d_{out}(w) = 0$ existiert. Wir nennen w die Wurzel des Graphen und unterscheiden zwischen *Out-Bäumen* und *In-Bäumen*, wobei bei einem Out-Baum jeder Knoten von der Wurzel und bei einem In-Baum die Wurzel von jedem Knoten aus erreicht werden kann. Wenn wir in dieser Arbeit von Bäumen im Bezug auf gerichtete Graphen sprechen, meinen wir immer Out-Bäume. In Wurzelbäumen ist y ein *Kind* von x und andersherum

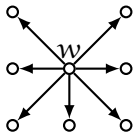
x der *Vater* von y, wenn die Kante (x, y) im Graphen vorkommt. Besitzt ein Knoten keine Kinder, so ist er ein *Blatt* des Graphen.

▶ Beispiel 5



Ein Beispiel für einen Wurzelbaum ist in der rechten Abbildung zu sehen. Der Graph besteht aus den Knotenmenge $V = \{w, a, b, c, d, e\}$ und der Kantenrelation $E^G = \{(w, a), (w, b), (w, c), (a, d), (a, e)\}$. Im Graphen ist der Knoten w die Wurzel mit den Kindern a, b und c, weiter ist der Knoten a der Vater von d und e. Die Knoten b, c, d und e sind die Blätter des Graphen. ◁

▶ Beispiel 6 (Stern)



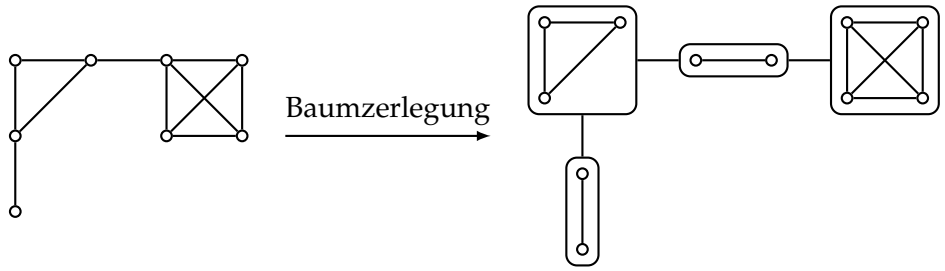
Eine spezielle Form von Wurzelbäumen sind *Sterne*. Bei diesen Graphen sind alle Knoten außer der Wurzel Blätter. Wir können Sterne nutzen, um eine natürliche Zahl n zu modellieren, indem wir einen Stern mit $n - 1$ Blättern zeichnen. In der Abbildung auf der rechten Seite ist ein Stern mit 7 Blättern zu sehen, er repräsentiert entsprechend die Zahl 8. ◁

Für die Betrachtungen in Abschnitt 2.4 benötigen wir die Begriffe der *Baumweite* und *Baumtiefe*. Die Baumweite $\mathrm{tw}(G)$ gibt die Ähnlichkeit eines Graphen G zu einem Baum an. Um diese Ähnlichkeit zu beschreiben, nutzen wir die Methode der *Baumzerlegung*. Für einen Graphen G, einen Baum T und eine Familie von Knotenmengen $\mathcal{V} = (V_t)_{t \in T}$ mit $V_t \subseteq V(G)$ heißt das Tupel $(T, \mathcal{V})$ eine *Baumzerlegung* von G, wenn folgende Eigenschaften gelten:

- ▷ $V(G) = \bigcup_{t \in T} V_t$,
- ▷ zu jeder Kante e gibt es einen Knoten t in T, so dass der Start- und Endknoten von e in der Knotenmenge V_t liegt,
- ▷ sind t_1, t_2, t_3 drei Knoten aus T, so muss $V_{t_1} \cap V_{t_3} \subseteq V_{t_2}$ gelten, wenn ein Pfad $t_1 \to \cdots \to t_2 \to \cdots \to t_3$ existiert.

Die *Weite* einer solchen Baumzerlegung ist die maximale Anzahl an Knoten in einer der Knotenmenge minus 1. Die Baumweite eines Graphen ist dann die kleinste Weite über alle seine Baumzerlegungen. Somit besitzen Bäume selbst eine Baumweite von 1.

► Beispiel 7 (Baumzerlegung)
In der folgenden Abbildung ist ein Graph und eine seiner Baumzerlegungen dargestellt. Diese Baumzerlegung besitzt die Weite 3.



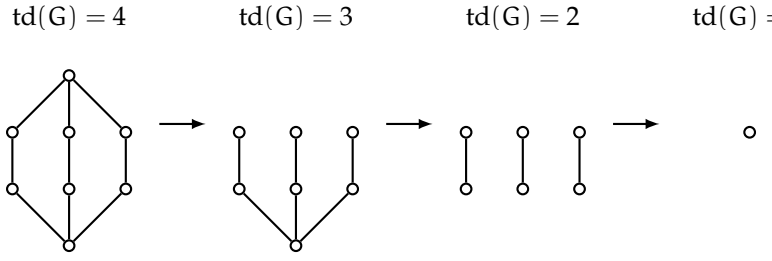
◁

Der von Nešetřil und Ossona de Mendez eingeführte Begriff der Baumtiefe soll entsprechend zum Begriff der Baumweite die Ähnlichkeit eines Graphen zu einem Stern beschreiben [10]. Für einen Graphen G mit Komponenten $G_1, \ldots, G_p$ ist die Baumtiefe $\mathrm{td}(G)$ definiert durch:

$$\mathrm{td}(G) = \begin{cases} 1 & |G(V)| = 1, \\ 1 + \min\limits_{v \in V(G)} \mathrm{td}(G - v) & p = 1 \text{ und } |G(V)| > 1, \\ \max\limits_{i=1}^{p} \mathrm{td}(G_i) & \text{sonst.} \end{cases}$$

In einem Wurzelbaum entspricht diese Definition dem längsten Pfad von der Wurzel zu einem Blatt. Ein Stern besitzt immer die Baumtiefe 2.

► Beispiel 8 (Baumtiefe)
Die folgende Abbildung zeigt die rekursive Berechnung der Baumtiefe eines Graphen. Bei dieser werden entsprechend der Vorschrift nach und nach Knoten entfernt und die berechnete Baumtiefe für jeden entfernten Knoten um 1 erhöht.



◁

2.3 Grundbegriffe der Platzkomplexität

In dieser Arbeit soll die Berechnungskomplexität von verschiedenen Varianten von UNARY-SUBSETSUM untersucht werden, dazu werden wir in diesem Abschnitt zunächst formal definieren, wie sich die Komplexität eines Problems ausdrückt. Die *Komplexität* eines Problems beschreibt den zum Lösen des Problems benötigten *Rechenaufwand*, der sich im Verbrauch von *Ressourcen* widerspiegelt. Man unterscheiden viele verschiedene Ressourcen; die wichtigsten sind *Zeit* und *Platz* – also die Anzahl der Berechnungschritte, die im Bezug zur Eingabe nötig sind, sowie die Anzahl der Speicherzellen, die während der Berechnung genutzt werden [7, 12]. Auch wenn unsere Computer immer schneller werden und über größeren Speicher verfügen, so werden auch die Probleminstanzen, die wir betrachten, immer größer. Daher ist es äußerst wichtig, effiziente Algorithmen zu entwickeln, die mit wenig Zeit und Platz auskommen. In dieser Arbeit werden wir uns hauptsächlich mit der Platzkomplexität beschäftigen. Es sei aber angemerkt, dass alle Platzklassen, die wir betrachten, in der Zeitklasse P liegen. Die Klasse P beschreibt dabei alle Probleme, die von einer Turingmaschine mit einer zur Eingabe polynomiellen Zeitschranke gelöst werden können.

Die erste Platzklasse, die wir nutzen wollen, ist die Klasse L – die Klasse der Probleme, die von Turingmaschinen unter Verwendung von logarithmischem Platz im Bezug zur Eingabelänge entschieden werden können.

Die Klasse NL beinhaltet alle Probleme, die von einer nichtdeterministischen Turingmaschine mit im Bezug zur Eingabelänge logarithmischem Platzverbrauch entschieden werden können. Das Erreichbarkeitsproblem in gerichteten Graphen ist ein Beispiel für ein wichtiges Problem, das vollständig für NL ist [12]. Wir werden einige Varianten von UNARY-SUBSETSUM ebenfalls als NL-vollständig klassifizieren.

Beide Klassen, L und NL, bauen auf dem Modell der Turingmaschine auf. Im Folgenden werden wir Platzklassen betrachten, die sich an *Schaltkreisen* orientieren. Dies ist sinnvoll, denn moderne Computer bestehen aus einer Vielzahl solcher Schaltkreise und die hier betrachteten Modelle haben daher einen hohen Bezug zur Praxis. Wir werden Schaltkreise formal als gefärbte Graphen einführen und uns dabei am Buch von Vollmer orientieren [16].

Eine grundlegende Eigenschaft von Schaltkreisen ist, dass ein Schaltkreis nur für eine bestimmte Eingabelänge konzipiert ist und entsprechend auch eine feste Ausgabelänge besitzt. Formal beschreibt ein Schaltkreis also für feste n und m eine Funktion $\varphi_C : \{0,1\}^n \to \{0,1\}^m$. Ein solcher Schaltkreis C ist eine logische Struktur

$$C = (V, E^C, I_1^C, \ldots, I_n^C, O_1^C, \ldots, O_m^C, G_\wedge^C, G_\vee^C, G_\neg^C, G_{\Theta_1}^C, \ldots, G_{\Theta_\infty}^C)$$

über der logischen Signatur

$$\tau_C = (E^2, I^1_1, \ldots, I^1_n, O^1_1, \ldots, O^1_m, G^1_\wedge, G^1_\vee, G^1_\neg, G^1_{\Theta_1}, \ldots, G^1_{\Theta_\infty}).$$

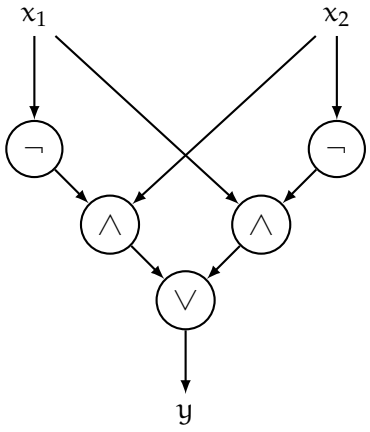
In der Struktur C ist V eine nichtleere Knotenmenge, die mit der Kantenrelation E^C einen gerichteten, zyklenfreien Graphen bildet. Die Relationen I^C_i weisen für $i \in \{1, \ldots, n\}$ die *Eingabeknoten* aus, entsprechend beschreiben die Relationen O^C_j mit $j \in \{1, \ldots, m\}$ die *Ausgabeknoten*. Durch die Relationen $G^C_\wedge, G^C_\vee, G^C_\neg, G^C_\Theta$ werden Knoten logische *Gatter* der Form $\{\wedge, \vee, \neg, \Theta\}$ zugewiesen, wobei ein Θ-Gatter ein *Threshold*-Gatter ist. Eine solche logische Struktur muss eine Vielzahl von Bedingungen erfüllen, um ein Schaltkreis zu sein. Daher muss für jeden Schaltkreis C gelten: $C \models \varphi_{\text{CIRC}}$, wobei die logische Formel φ_{CIRC} für $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m\}$ und $k \in \{\wedge, \vee, \neg, \Theta\}$ beschreibt, dass:

- ▷ der Graph (V, E^C) zyklenfrei ist;
- ▷ jeder Knoten in genau einer Relation I^C_i, O^C_j, G^C_k liegt;
- ▷ jede Relation I^C_i und O^C_j genau ein Element enthält;
- ▷ für Knoten $x \in V$, die in einer Relation I^C_i liegen, gilt $d_{\text{in}}(x) = 0$;
- ▷ für Knoten $x \in V$, die in einer Relation O^C_j liegen, gilt $d_{\text{in}}(x) = 1$ und $d_{\text{out}}(x) = 0$.

Stellen wir einen Schaltkreis als Graphen dar, so schreiben wir in die Knoten, die ein Gatter repräsentieren, den entsprechenden logischen Operator. Für Knoten, die für ein Ein- oder Ausgabebit stehen, notieren wir nur die Nummer des Bits.

Die Auswertung eines Schaltkreises geschieht rekursiv. Der Wahrheitswert eines Ausgabebits entspricht jenem des Vaters. Der Wahrheitswert eines $\{\wedge, \vee, \neg\}$-Gatters wird entsprechend der logischen Operation und der Wahrheitswerte seiner Väter ausgewertet. Ein Θ_k-Gatter wird zu wahr ausgewertet, wenn mindestens k seiner Väter zu wahr ausgewertet werden. Die Wahrheitswerte von Eingabeknoten werden gesetzt.

► Beispiel 9 (XOR)
Der folgende Schaltkreis berechnet die XOR-Funktion zweier Bits x_1 und x_2:



◁

Ein Problem des Schaltkreismodells ist, dass jeder Schaltkreis nur Eingaben einer festen Länge verarbeiten kann. Beschreiben wir Schaltkreise, die ein gegebenes Problem entscheiden, so fassen wir daher Schaltkreise für alle Eingabelängen zu einer *uniformen Familie* von Schaltkreisen zusammen. Für ein Problem existiert eine uniforme Familie von Schaltkreisen, wenn eine Turingmaschine existiert, die zu einer Eingabe $n \in \mathbb{N}$ den Schaltkreis ausgibt, der für das Problem alle Eingaben der Länge n entscheidet. Das Problem dieser Definition ist die Mächtigkeit der Turingmaschine, denn diese könnte das Problem selbst lösen und die Lösung in den Schaltkreis kodieren. Für ein für die Komplexitätstheorie sinnvolles Rechenmodell müssen wir jedoch eine Familie von Schaltkreisen beschreiben, die das Problem selbst berechnen kann. Um die Mächtigkeit der Turingmaschine zu beschränken, betrachten wir in dieser Arbeit daher nur DLOGTIME-uniforme Familien von Schaltkreisen. Zu einer solchen Familie von Schaltkreisen existiert, vereinfacht gesagt, eine deterministische Turingmaschine, die zu einer Eingabe $n \in \mathbb{N}$ unter einer $\mathcal{O}(\log(n))$-Zeitschranke den Schaltkreis ausgibt, der alle Eingaben der Länge n entscheiden kann. Für eine genaue Beschreibung von verschiedenen Arten von uniformen Familien von Schaltkreisen verweisen wir an dieser Stelle auf Ruzzo [14].

Im Folgenden stellen wir drei Arten von Schaltkreisen und die dazugehörigen Komplexitätsklassen vor. Die erste Familie von Schaltkreisen, die wir betrachten, sind die NC-Schaltkreise. Ein NC^i-Schaltkreis ist ein Schaltkreis, der nur aus Ein- und Ausgabeknoten sowie den Gattern $\{\wedge, \vee, \neg\}$ besteht. Die Gatter eines NC^i-Schaltkreises besitzen maximal zwei eingehende Kanten. Ein NC^i-Schaltkreis, der eine Eingabe der Länge $n \in \mathbb{N}$ verarbeitet, darf im Bezug zu n nur polynomiell viele Gatter besitzen und seine Tiefe muss durch $\mathcal{O}(\log^i(n))$ beschränkt sein. Die Komplexitätsklasse NC^i ist nun defi-

niert als die Menge aller Probleme, für die eine DLOGTIME-uniforme Familie von NC^i-Schaltkreises existiert, die diese entscheiden.

Betrachten wir einen NC^i-Schaltkreis und erlauben den Gattern beliebig viele eingehende Kanten, so erhalten wir einen AC^i-Schaltkreis. Die Komplexitätsklasse AC^i ist dann analog zur Komplexitätsklasse NC^i definiert. Eine wichtige AC^i-Klasse ist zum Beispiel AC^0, denn diese beschreibt gerade die Probleme, die sich durch logische Formeln erster Stufe ausdrücken lassen [8].

Erweitern wir einen AC^i-Schaltkreis um Θ-Gatter, so erhalten wir einen sogenannten TC^i-Schaltkreis. Die Komplexitätsklasse TC^i definiert sich analog zu den Komplexitätsklassen NC^i und AC^i. Für diese Arbeit ist die Klasse TC^0 von Interesse, denn wir werden UNARY-SUBSETSUM und einige seiner Varianten als TC^0-vollständig klassifizieren. Ein anderes bekanntes Beispiel für ein TC^0-vollständiges Problem ist MAJORITY [16].

► Problem 10 (Majority)
Eingabe: Ein Vektor $\vec{b} \in \{0,1\}^p$ mit $p \in \mathbb{N}$.
Frage: Sind in $\vec{b}$ mindestens die Hälfte der Elemente 1?
Sprache: $\text{MAJORITY} = \{ w \in \{0,1\}^\star \mid \#_1(w) \geqslant |w|/2 \}$. ◁

Zwischen den vorgestellten Komplexitätsklassen sind die folgenden Zusammenhänge bekannt [16]: ein Gatter mit beliebig vielen eingehenden Kanten kann durch einen Schaltkreis logarithmischer Tiefe, der nur Gatter mit je zwei eingehenden Kanten besitzt, simuliert werden. Dies führt uns zu $NC^i \subseteq AC^i \subseteq NC^{i+1}$. Auch ein Θ-Gatter kann durch einen Schaltkreis logarithmischer Tiefe, der keine Θ-Gatter enthält, simuliert werden. Dies impliziert $AC^i \subseteq TC^i \subseteq AC^{i+1}$. Weiter ist für Schaltkreise und die Klassen L und NL der Zusammenhang $NC^1 \subseteq L \subseteq NL \subseteq AC^1$ bekannt und wir können daher insgesamt Folgendes formulieren:

$$NC^0 \subseteq AC^0 \subseteq TC^0 \subseteq NC^1 \subseteq L \subseteq NL \subseteq AC^1 \subseteq TC^1 \subseteq NC^2 \subseteq \cdots \subseteq P.$$

2.4 Die Logspace-Versionen der Sätze von Bodlaender und Courcelle

Wir betrachten nun die Werkzeuge, die nötig sind um Aussagen über die Komplexität von in monadischer Logik zweiter Stufe formulierten Problemen über Graphen mit beschränkter Baumweite oder Baumtiefe zu treffen. Dabei wollen wir die von Elberfeld, Jakoby und Tantau formulierte Version der Sätze von Bodlaender und Courcelle nutzen. Der Satz von Bodlaender zeigt, dass es in linearer Zeit möglich ist, die Baumzerlegung eines Graphen mit beschränkter Baumweite zu berechnen [1]. Damit zeigt der Satz von

Courcelle, dass auch in monadischer Logik zweiter Stufe formulierte Aussagen über Graphen mit beschränkter Baumweite in linearer Zeit entschieden werden können [2]. Elberfeld, Jakoby und Tantau zeigten, dass in beiden Sätzen die lineare Zeitschranke durch eine logarithmische Platzschranke ersetzt werden kann und formulierten so den folgenden Satz [4].

► Satz 11
Für eine feste Signatur τ existiert für jedes $k \geqslant 1$ und jede in monadischer Logik zweiter Stufe formulierten Formel $\varphi(X_1, \ldots, X_n)$ eine Logspace-Turingmaschine, die zu jeder logischen Struktur S über τ, mit einer durch k beschränkten Baumweite, das Lösungshistogramm $\text{histogram}(S, \varphi)$ berechnet. □

Weiter konnten Elberfeld, Jakoby und Tantau eine weitere Variante des Satzes formulieren, indem die beschränkte Baumweite durch eine beschränkte Baumtiefe ersetzt wird [5].

► Satz 12
Für eine feste Signatur τ existiert zu jedem $d \in \mathbb{N}$ und jeder in monadischer Logik zweiter Stufe formulierten Formel $\varphi(X_1, \ldots, X_\ell, Y_1, \ldots, Y_k)$ eine DLOGTIME-uniforme Familie von TC^0-Schaltkreisen, die zu einer logischen Struktur S über τ mit durch d beschränkter Baumtiefe, einem Index s mit ℓ Dimensionen und einer Bitposition i das Bit an Position i des Lösungshistogramms $\text{histogram}(S, \varphi)[s]$ berechnet. □

Somit haben wir Werkzeuge, um zu zeigen, dass wir ein Problem unter Verwendung von logarithmischem Platz oder mit einem TC^0-Schaltkreis entscheiden können, wenn wir die Eingabe für dieses Problem als Graph mit beschränkter Baumweite oder Baumtiefe modellieren können und sich die Problemstellung auf eine MSO-Formel über diesem Graphen zurückführen lässt.

3 Bekannte Komplexitätsaussagen von UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM

In diesem Abschnitt betrachten wir bekannte Komplexitätsaussagen von UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM. Dabei werden wir zunächst auf die TC^0-Vollständigkeit von UNARY-SUBSETSUM eingehen, die aus dem Satz von Elberfeld, Jakoby und Tantau folgt [5]. Im zweiten Teil des Abschnittes betrachten wir die Komplexität von PARTIAL-UNARY-SUBSETSUM und präsentieren eine überarbeitete Version des Beweises von Moniens Satz zur NL-Vollständigkeit [9].

3.1 Die Komplexität von UNARY-SUBSETSUM

In diesem Abschnitt untersuchen wir die Komplexität von UNARY-SUBSETSUM. Dazu nutzen wir die Sätze aus Abschnitt 2.4, um UNARY-SUBSETSUM $\in TC^0$ zu zeigen. Anschließend reduzieren wir von MAJORITY, um die TC^0-Härte zu folgern.

► Satz 13
Das Problem UNARY-SUBSETSUM ist TC^0-vollständig.
Beweis: Wir betrachten eine Instanz von UNARY-SUBSETSUM der Form

$$0^{a_1}10^{a_2}10\ldots10^{a_p}10^{s}.$$

Zu dieser konstruieren wir einen ungerichteten Graphen $G = (V, E^G)$, indem wir für jeden Wert a_i mit $i \in \{1,\ldots,p\}$ einen Stern mit a_i Knoten erzeugen. Diese Konstruktion gelingt, denn da die Werte a_i unär kodiert sind, kann ein TC^0-Schaltkreis die Eingabe lesen und entsprechende Knoten und Kanten ausgeben. Wir betrachten nun die MSO-Formel $\varphi(X)$, für die G genau dann ein Modell ist, wenn X Sterne entweder vollständig oder gar nicht enthält. Dabei enthält X einen Stern vollständig, wenn für alle Knoten v des Sterns $X(v)$ gilt. Die Formel $\varphi(X)$ sei wie folgt definiert:

$$\varphi(X) = \forall x \,.\, X(x) \rightarrow \forall y \,.\, E(x,y) \rightarrow X(y).$$

Die Baumtiefe eines Sterns ist konstant 2 und somit beschränkt, folglich können wir nach Satz 12 für alle Strukturen G dieser Art das Lösungshistogramm $\text{histogram}(G, \varphi)$ in TC^0 berechnen. Dabei ist das Lösungshistogramm an einem Index ν größer 0, wenn Sterne so gewählt werden können, dass die Summe ihrer Knoten ν ergibt. Folglich muss ein TC^0- Schaltkreis nur prüfen, ob $\text{histogram}(G, \varphi)[s] > 0$ gilt. Dazu berechnen mehre TC^0-Schaltkreise je ein Bit von $\text{histogram}(G, \varphi)[s]$ und die Ausgaben dieser Schaltkreise münden in einem Θ-Gatter mit $k = 1$. Wird dieses Θ-Gatter zu wahr ausgewertet, so ist der Eintrag $\text{histogram}(G, \varphi)[s] > 0$ und die Instanz ist lösbar. Folglich können wir UNARY-SUBSETSUM mit einem TC^0-Schaltkreis entscheiden.

Um die TC^0-Härte von UNARY-SUBSETSUM zu zeigen, führen wir im Folgenden eine FO(BIT)-Reduktion von MAJORITY auf UNARY-SUBSETSUM durch. Als Instanz von MAJORITY betrachten wir einen Binärstring $x \in \{0,1\}^n$, gegeben als logische Struktur S, die wie bekannt einen Binärstring beschreibt und definiert ist durch:

$$S = \big(U = \{1, 2, \ldots, n\},\ \leqslant^S,\ I_1^S = \{i_1, \ldots, i_r\} \big).$$

Aus diesem Binärstring erzeugen wir den Binärstring $S' = (U', \leqslant'^{S'}, I_1'^{S'})$, der als Eingabe für UNARY-SUBSETSUM dienen kann. Damit die Stringlänge keine Aussage über die Anzahl der 1*en* in der MAJORITY-Instanz liefert, bilden wir eine 0 der MAJORITY-Instanz auf 11 und eine 1 auf 01 ab. Damit wir zusätzlich Platz für die Zielsumme haben, müssen wir daher die Universumsgröße verdreifachen und setzen dazu:

$$U' = \{\, (x_1, x_2) \mid S \models \varphi_0(x_1, x_2) \,\}.$$

Um $\varphi_0(x_1, x_2)$ zu beschreiben, benötigen wir zunächst die folgenden vier Formeln:

- $\text{PRE}(x, y) = (x \neq y) \wedge (x \leqslant y)$
  $\wedge \neg\exists z \,.\, (z \neq x) \wedge (z \neq y) \wedge (x \leqslant z) \wedge (z \leqslant y)$, wird zu wahr ausgewertet, wenn x der direkte Vorgänger von y ist;
- $\text{FIRST}(x) = \neg\exists y \,.\, \text{PRE}(y, x)$ wird zu wahr ausgewertet, wenn x das kleinste Element des Universums ist;
- $\text{SECOND}(x) = \exists y \,.\, \text{FIRST}(y) \wedge \text{PRE}(y, x)$ wird zu wahr ausgewertet, wenn x das zweit kleinste Element des Universums ist;
- $\text{THIRD}(x) = \exists y \,.\, \text{SECOND}(y) \wedge \text{PRE}(y, x)$ wird zu wahr ausgewertet, wenn x das dritt kleinste Element des Universums ist;

Nun lässt sich $\varphi_0(x_1, x_2)$ definieren durch:

$$\varphi_0(x_1, x_2) = \text{FIRST}(x_2) \vee \text{SECOND}(x_2) \vee \text{THIRD}(x_2).$$

Um in diesem Universum Elemente auf 1 zu setzen, müssen wir nun die Ordnung definieren, dazu setzen wir:

$$\leqslant'^{S'} = \{\, (x_1, x_2, y_1, y_2) \mid (\neg \textsc{third}(x_2) \wedge \textsc{third}(y_2)) \\ \vee\, (x_1 \leqslant y_1) \wedge (x_1 = x_2 \rightarrow x_2 \leqslant y_2) \,\}.$$

Dadurch besteht das Universum aus zwei Blöcken, wobei der erste die doppelte und der zweite die Größe des Ausgangsuniversums besitzt Der erste Block ist groß genug um alle Zahlen der MAJORITY-Instanz zu codieren, dies geschieht durch die folgende Formel:

$$\varphi_1(x_1, x_2) = \neg I_1(x_1) \vee \textsc{second}(x_2).$$

Der zweite Block ist allein der Zielsumme der UNARY-SUBSETSUM-Instanz vorbehalten. Da wir MAJORITY lösen wollen, muss hier also die hintere Hälfte der Zahlen 0 sein, was einer Zielsumme der halben Stringlänge entspricht. Alle anderen Zahlen setzen wir auf 1 (was in der UNARY-SUBSETSUM-Instanz einer Reihe von vielen 0*en* entspricht). Um dies zu beschreiben benötigen wir zwei weitere logische Formeln:

- ▷ $\textsc{unisize}(x) = \neg\exists y \,.\, (x \neq y) \wedge (x \leqslant y)$ wird zu wahr ausgewertet, wenn x die Größe des Universums ist;
- ▷ $\textsc{half}(x, y) = \textsc{add}(x, x, y) \vee \exists z \,.\, \textsc{pre}(z, y) \wedge \textsc{add}(x, x, z)$ wird zu wahr ausgewertet, wenn x die Hälfte von y ist.

Damit lässt sich nun der zweite Block durch folgende Formel beschreiben:

$$\varphi_2(x_1, x_2) = \textsc{third}(x_2) \wedge \exists z y \,.\, \textsc{unisize}(z) \wedge \textsc{half}(y, z) \wedge x_1 \leqslant y.$$

Nun setzen wir $I_1'^{S'}(x_1, x_2)$ auf:

$$I_1'^{S'}(x_1, x_2) = \{\, (x_1, x_2) \mid S \models \varphi_1(x_1, x_2) \vee \varphi_2(x_1, x_2) \,\}$$

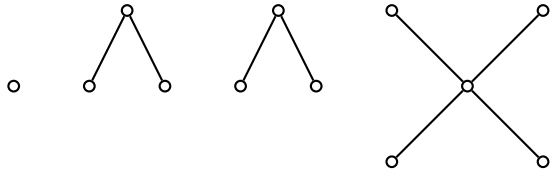
und erhalten dadurch einen wie oben beschriebenen String, der als Eingabe einer UNARY-SUBSETSUM-Instanz dient.

Wir haben somit aus einer Instanz $x \in \{0,1\}^n$ von MAJORITY eine Instanz von UNARY-SUBSETSUM konstruiert. Es bleibt zu zeigen, dass diese Instanz genau dann lösbar ist, wenn auch die Instanz von MAJORITY lösbar ist. Der Wertevektor $\vec{a}$ der UNARY-SUBSETSUM-Instanz besteht nur aus 0*en* und 1*en*, weshalb jede Zielsumme, die kleiner gleich dem Gewicht von $\vec{a}$ ist, erreicht werden kann. Weiter enthält $\vec{a}$ für jedes Bit in x eine 0 oder 1, je nachdem, welchen Wert das entsprechende Bit in x besitzt. Die Zielsumme der UNARY-SUBSETSUM-Instanz ist $s = \lfloor n/2 \rfloor$ und lässt sich entsprechend nur erreichen, wenn $\vec{a}$ mindestens $s = \lfloor n/2 \rfloor$ viele 1*en* enthält (Achtung: in der Kodierung entsprechen diese 0*en*). Ist die Instanz von MAJORITY lösbar, so ist das Gewicht von x größer gleich $s = \lfloor n/2 \rfloor$ und folglich enthält $\vec{a}$ genügend

viele 1*en* und die Instanz von UNARY-SUBSETSUM ist lösbar. Ist wiederum MAJORITY nicht lösbar, so ist das Gewicht von x kleiner als $s = \lfloor n/2 \rfloor$ und entsprechend enthält auch $\vec{a}$ nicht genügend viele 1*en* um die Zielsumme s zu erreichen. □

▶ Beispiel 14

Wir betrachten eine Instanz von UNARY-SUBSETSUM mit $\vec{a} = (1, 3, 3, 5)$ und $s = 6$. Zu dieser Instanz konstruieren wir einen Graphen aus Sternen, wie in der folgenden Abbildung zu sehen.



Die folgende Tabelle zeigt das Lösungshistogramm der UNARY-SUBSETSUM-Instanz, das wir zu dem oben abgebildeten Graphen berechnet haben.

$\|X\|$	1	2	3	4	5	6	7	8	9	10	11	12
Anzahl der Lösungen	1	0	2	2	0	2	1	2	2	0	1	1

Wir können aus Spalte 6 entnehmen, dass für die Zielsumme s eine Lösung existiert, folglich ist diese Instanz von UNARY-SUBSETSUM lösbar. Aus dem Histogramm folgt auch, dass diese Instanz für beispielsweise $s = 10$ nicht lösbar wäre. ◁

▶ Beispiel 15

Folgendes Beispiel soll die Reduktion von MAJORITY auf UNARY-SUBSETSUM veranschaulichen. Sei dazu $x = 001101$ eine Instanz von MAJORITY. Der erste Block der UNARY-SUBSETSUM-Instanz ergibt sich, indem wir die 0*en* durch 11 und die 1*en* durch 01 ersetzen, wir erhalten also 111101011101. Den zweiten Block erhalten wir durch einen String der Länge des Ausgangswortes, indem wir die erste Hälfte der Zahlen 1 setzen, also 111000. Zusammen bildet dann 111101011101111000 die Eingabe für UNARY-SUBSETSUM. ◁

3.2 Die Komplexität von PARTIAL-UNARY-SUBSETSUM und der Satz von Monien

In diesem Abschnitt betrachten wir Moniens NL-Vollständigkeitsbeweis für PARTIAL-UNARY-SUBSETSUM [9]. Da der Beweis von Monien sehr technisch und schwer verständlich ist, diskutieren und überarbeiten wir ihn ausführlich.

Zunächst zeigen wir, dass PARTIAL-UNARY-SUBSETSUM von einer nichtdeterministischen Turingmaschine mit logarithmischer Platzbeschränkung entschieden werden kann und anschließend, dass dieses Problem NL-hart ist.

▶ Lemma 16
Es gilt PARTIAL-UNARY-SUBSETSUM $\in$ NL.

Beweis: Betrachten wir eine Instanz von PARTIAL-UNARY-SUBSETSUM der Form

$$0^{a_1}10^{c_1}110^{a_2}10^{c_2}11\ldots110^{a_p}10^{c_p}1110^{b}.$$

Eine nichtdeterministische Logspace-Turingmaschine liest diese Eingabe von links nach rechts und rät dabei einige a_i, die sie dann auf ihrem Arbeitsband aufsummiert. Während sie über die Eingabe läuft, prüft sie jeweils, ob der Wert auf ihrem Arbeitsband größer gleich dem aktuellen c_i auf dem Eingabeband ist und verwirft sofort, wenn dies nicht der Fall ist. Am Ende prüft sie, ob die Summe auf ihrem Arbeitsband gleich b ist und akzeptiert gegebenenfalls. Da die Eingabe unär ist, kann die Summe unter Verwendung von logarithmischem Platz berechnet werden und es folgt die Aussage. □

Um nun die NL-Härte von PARTIAL-UNARY-SUBSETSUM zu zeigen, reduzieren wir von einer unäre Instanz des Erreichbarkeitsproblems in Graphen – von dem bekannt ist, dass es NL-vollständig ist [12] – auf eine Instanz von PARTIAL-UNARY-SUBSETSUM. Dabei betrachten wir die folgende Variante des Problems:

▶ Problem 17 (Erreichbarkeitsproblem in gerichteten, zyklenfreien Graphen)
Eingabe: Ein zyklenfreier, gerichteter Graph $G = (V, E^G)$ mit Knotenmenge $V = \{1, \ldots, q\}$.
Frage: Gibt es in G einen Pfad von 1 nach q?
Sprache: DAG-REACH $= \{\, G = (V, E^G), V = \{1, \ldots, q\} \mid$
G ist ein zyklenfreier, gerichteter Graph und
es existiert ein Pfad von 1 nach q $\}$. ◁

Für diese Reduktion von DAG-REACH auf PARTIAL-UNARY-SUBSETSUM benötigen wir zunächst noch das folgende Lemma:

▶ Lemma 18
Für r beliebige natürliche Zahlen $1 \leqslant p_1 \leqslant p_2 \leqslant \cdots \leqslant p_r$ besitzt das folgende System keine Lösung im $\mathbb{Z}^r$, wenn für einige ν gilt $p_\nu = p_{\nu+1}$. Andernfalls

besitzt es nur die Lösung $\vec{x} = \vec{0}$.

$$\sum_{i=1} x_i = 0 \tag{i}$$

$$\sum_{i=1}^{r} \left(p_i x_i + \frac{x_i(x_i+1)}{2} \right) = 0 \tag{ii}$$

$$\sum_{i=1}^{\nu} \left(p_i x_i + \frac{x_i(x_i+1)}{2} \right) \geqslant 0 \quad \text{für alle } \nu \leqslant r \tag{iii}$$

$$x_{\nu+1} > p_\nu - p_{\nu+1} \quad \text{für alle } \nu \leqslant r \tag{iv}$$

Beweis: Zunächst müssen wir zeigen, dass $\vec{x} = \vec{0}$ eine Lösung ist, wenn für kein ν gilt, dass $p_\nu = p_{\nu+1}$. Die Gleichungen (i) – (iii) werden durch $\vec{x} = \vec{0}$ gelöst, denn alle Summanden sind somit 0. Ein $x_{i+1} = 0$ löst die vierte Gleichung nur, wenn $p_i \neq p_{i-1}$ gilt, da sonst $0 > 0$ gefordert wird. Es folgt dieser Teil der Aussage.

Es bleibt zu zeigen, dass keine andere Lösung existiert. Dazu nehmen wir an, dass eine Lösung existiert, in der einige $x_i \neq 0$ existieren und zeigen, dass dies zu einem Widerspruch führt. Es sei x_q das kleinste $x_i \neq 0$. Aus (iv) folgt $x_q > -p_q$, denn es gilt $x_q > p_{q-1} - p_q \in [1 - p_q, 0]$, und $x_{\nu+1} > 0$ für $p_\nu = p_{\nu+1}$. Da x_q das kleinste $x_i \neq 0$ ist, folgt

$$\begin{aligned} \sum_{i=1}^{q} \left(p_i x_i + \frac{x_i(x_i+1)}{2} \right) &= p_q x_q + \frac{x_q(x_q+1)}{2} \\ &= \frac{x_q}{2}(2p_q + x_q + 1) \\ &\overset{(iii)}{\geqslant} 0. \end{aligned}$$

Aus $x_q > -p_q$ folgt $2p_q + x_q + 1 > 0$ und damit $x_q \geqslant 0$, denn sonst wäre der Term insgesamt negativ. Mit $x_q \neq 0$ erhalten wir $x_q > 0$. Wir können $q = 1$ annehmen, denn alle x_i mit $i < q$ sind 0.

Es sei $\alpha = x_1$ und $a = p_1 x_1 + \frac{x_1(x_1+1)}{2}$. Das System lässt sich dann formulieren als

$$a + \sum_{i=2}^{r} \left(p_i x_i + \frac{x_i(x_i+1)}{2} \right) = 0, \tag{1}$$

$$a + \sum_{i=2}^{\nu} \left(p_i x_i + \frac{x_i(x_i+1)}{2} \right) \geqslant 0 \quad \text{für all } \nu \leqslant r,$$

$$x_{\nu+1} \geqslant p_\nu - p_{\nu+1} \quad \text{für all } \nu \leqslant r,$$

$$\alpha + \sum_{i=2}^{r} x_i = 0. \tag{2}$$

Wir müssen nun zeigen, dass dieses System für beliebige $\alpha > 0$ keine Lösung im $\mathbb{Z}^{r-1}$ besitzt. Hierfür zeigen wir, dass Teilsystem (2) verletzt wird, wenn $\alpha > 0$ und

$$a \leqslant \alpha p_1 + \frac{\alpha(\alpha+1)}{2}$$

gilt, was zur Lösung des Systems zwingend notwendig wäre. Dazu führen wir eine Induktion über die Anzahl m der Summanden in $\sum x_i$ durch und zeigen, dass sich jedes System mit $m = r$ Summanden auf ein System mit $m = r-1$ Summanden zurückführen lässt.

IA: Mit $a = p_1 x_1 + \frac{x_1(x_1+1)}{2}$ und $\alpha > 0$ folgt die Aussage für $m = 0$.

IV: Für beliebige, aber feste m gilt, dass $\alpha + \sum_{i=2}^{r} x_i > 0$ aus $\alpha > 0$ und $a \leqslant \alpha p_1 + \frac{\alpha(\alpha+1)}{2}$ folgt.

IS: Die Aussage ist für $m = r-1$ gezeigt und wir betrachten nun den Fall $m = r$, wir wollen also ein System mit r Summanden auf eines mit $r-1$ Summanden zurückführen. Dazu verschmelzen wir α und x_2 zu β und a mit dem ersten Summanden zu b. Anschließend zeigen wir $\beta > 0$ und

$$b \leqslant \beta p_2 + \frac{\beta(\beta+1)}{2},$$

um ein gültiges neues System zu erhalten, für welches Teilsystem (2) ebenfalls verletzt wird.
Wir setzen $b = a + p_2 x_2 + \frac{x_2(x_2+1)}{2}$, womit sich Teilsystem (1) formulieren lässt als

$$\begin{aligned}
&b + \sum_{i=3}^{r+2} \left(p_i x_i + \frac{x_i(x_i+1)}{2} \right) = 0, \\
&b + \sum_{i=3}^{\nu} \left(p_i x_i + \frac{x_i(x_i+1)}{2} \right) \geqslant 0 \quad \text{für alle } \nu \leqslant r+2, \\
&\qquad\qquad x_{\nu+1} > p_\nu - p_{\nu-1} \quad \text{für alle } \nu \leqslant r+2.
\end{aligned}$$

Wir unterscheiden nun die Fälle $x_2 \geqslant 0$ und $x_2 < 0$. Es sei zunächst $x_2 \geqslant 0$ und wir setzen $\beta = \alpha + x_2$ und können b wegen $p_1 \leqslant p_2$ abschätzen durch

$$\begin{aligned}
b &\leqslant \alpha p_1 + \frac{\alpha(\alpha+1)}{2} + x_2 p_2 + \frac{x_2(x_2+1)}{2} \\
&\leqslant (\alpha + x_2) p_2 + \frac{(\alpha + x_2)(\alpha + x_2 + 1)}{2} \\
&= \beta p_2 + \frac{\beta(\beta+1)}{2}.
\end{aligned}$$

Da weiter $\beta = (\alpha + x_2) > 0$ gilt, folgt aus der Induktionsvoraussetzung, dass

$$\beta + \sum_{i=3}^{r+2} x_i > 0.$$

Wir betrachten nun den Fall $x_2 < 0$ und setzen $\gamma = -x_2 > 0$ und

$$b = a - \gamma p_2 + \frac{\gamma(\gamma-1)}{2} \overset{(iii)}{\geqslant} 0.$$

Dann liefert uns $p_1 \leqslant p_2$ und $\alpha, \gamma > 0$ die Ungleichungen

$$\alpha p_1 + \frac{\alpha(\alpha+1)}{2} \geqslant a \geqslant \gamma p_2 - \frac{\gamma(\gamma-1)}{2}.$$

Wir wollen nun $\alpha > \gamma$ zeigen und nehmen für einen Widerspruch $\gamma \geqslant \alpha$ an. Dann erhalten wir für die obigen Ungleichungen

$$\begin{aligned} \alpha p_1 + \frac{\alpha(\alpha+1)}{2} &\geqslant \gamma p_2 - \frac{\gamma(\gamma-1)}{2} \\ \Leftrightarrow \frac{\alpha(\alpha+1)}{2} + \frac{\gamma(\gamma-1)}{2} &\geqslant \gamma p_2 - \alpha p_1, \end{aligned}$$

und damit

$$\begin{aligned} \gamma^2 = \frac{\gamma(\gamma+1)}{2} + \frac{\gamma(\gamma-1)}{2} &\\ \geqslant \gamma p_2 - \alpha p_1 &\\ \geqslant \gamma p_2 - \gamma p_1 &\\ = \gamma(p_2 - p_1). & \end{aligned}$$

Dies ist ein Widerspruch zu $\gamma < p_2 - p_1$ bzw. $x_2 > p_1 - p_2$ und folglich gilt $\alpha > \gamma$. Wir setzen nun $\beta = \alpha - \gamma > 0$ und schätzen b ab durch

$$\begin{aligned} b &\leqslant a - \gamma p_2 + \frac{\gamma(\gamma-1)}{2} \\ &= \alpha p_1 - \gamma p_2 + \frac{\alpha(\alpha+1)}{2} + \frac{\gamma(\gamma-1)}{2} \\ &= \alpha p_2 - \gamma p_2 - \alpha p_2 + \alpha p_1 + \frac{\alpha^2 + \alpha + \gamma^2 - \gamma}{2} \\ &= (\alpha - \gamma) p_2 - \alpha(p_2 - p_1) + \frac{\alpha^2 - 2\alpha\gamma + \alpha + \gamma^2 - \gamma}{2} + \alpha\gamma \\ &= (\alpha - \gamma) p_2 - \alpha(p_2 - p_1) + \frac{(\alpha-\gamma)(\alpha-\gamma+1)}{2} + \alpha\gamma \\ &\leqslant (\alpha - \gamma) p_2 + \frac{(\alpha-\gamma)(\alpha-\gamma+1)}{2} \\ &= \beta p_2 + \frac{\beta(\beta+1)}{2}. \end{aligned}$$

Mit der Induktionsvoraussetzung folgt dann

$$\beta + \sum_{i=3}^{r+2} x_i > 0.$$

Wir haben gezeigt, dass $\vec{0}$ eine Lösung des Systems ist, wenn $p_\nu = p_{\nu+1}$ für kein ν gilt, und dass es keine andere Lösung gibt. Entsprechend gilt die Behauptung. □

Mithilfe dieses Lemmas können wir nun den eigentlichen Satz formulieren.

▶ Satz 19
Das Problem PARTIAL-UNARY-SUBSETSUM ist NL-hart.

Beweis: Wir reduzieren von DAG-REACH auf PARTIAL-UNARY-SUBSETSUM; dazu betrachten wir eine Instanz von DAG-REACH in der Form eines gerichteten, zyklenfreien Graphen $G = (V, E^G)$ mit Knoten $V = (1, \ldots, q)$ und Kanten $(i, j) \in E(G)$ mit $i < j$. Weiter seien $\{1, \ldots, r_i\}$ die Nachbarn von Knoten i und $\delta^i_j = j - i$ der Abstand der Kante (i, j), welcher wegen $i < j$ immer positiv ist.

Wir konstruieren zu einer solchen Instanz nun eine Instanz des Problems PARTIAL-UNARY-SUBSETSUM. Dazu bestimmen wir für Knoten und Kanten verschiedene a_i und beschreiben $\vec{c}$ und b so, dass wir alle a_i, die zu einem Knoten korrespondieren, wählen müssen. Dabei wollen wir die Instanz so gestalten, dass die Wahl der Knoten dafür sorgt, dass zwingend auch Kanten gewählt werden müssen, um in der Summe die c_i bzw. b zu erreichen. Dazu werden wir Zahlen d_1 und d_2 einführen, die sich gegenseitig nicht beeinflussen können. Dadurch können wir genau unterscheiden, ob Knoten oder Kanten gewählt werden. Weiter sorgen wir dafür, dass nur die Wahl bestimmter Kanten die Zielsumme b erzeugen kann und dass diese Kanten jene sind, die im Graphen einen Pfad von 1 nach q bilden. Um die Korrektheit unserer Konstruktion zu beweisen, werden wir anschließend zeigen, dass diese Instanz lösbar ist, wenn im Graphen ein Pfad von 1 nach q existiert und dass andersherum ein solcher Pfad existiert, wenn die Instanz eine Lösung besitzt.

Für die Instanz von PARTIAL-UNARY-SUBSETSUM bestimmen wir für jeden Knoten $1, i \in V$ mit $i > 1$ einen Wert

$$\begin{aligned} a_{10} &= 0, \\ a_{i0} &= d_2 - i d_1. \end{aligned}$$

Weiter berechnen wir für jede Kante (i, j) den Wert

$$a_{ij} = \left(i\delta^i_j + \frac{\delta^i_j(\delta^i_j + 1)}{2} \right) d_1 + \delta^i_j.$$

Nun wollen wir d_1 und d_2 so bestimmen, dass sie sich gegenseitig nicht beeinflussen und dass Terme, die nicht mit d_1 oder d_2 faktorisiert sind, solche die es sind nicht beeinflussen können. Es gilt $a_{ij} = \alpha_{ij} d_1 + \beta_{ij}$ mit

$$\alpha_{ij} \leqslant i(q-i) + \frac{(q-i)(q-i+1)}{2} = \frac{(q^2 - i^2 + q - i)}{2} < q^2,$$

da $i > 0$ und $\beta_{ij} \leqslant (q-1)$. Wir wählen nun $d_1 = q^3$ und $d_2 = q^7$, dann ist die Summe über alle β_{ij} durch $d_1 - 1$ beschränkt, denn der Graph kann maximal q^2 Kanten besitzen, entsprechend gilt

$$\begin{aligned}\sum_{i=1}^{q^2} \beta_{ij} &\leqslant \sum_{i=1}^{q^2} (q-1) \\ &= q^2(q-1) \\ &= q^3 - q^2 \\ &\leqslant q^3 - 1 \\ &= d_1 - 1.\end{aligned}$$

Weiter ist die Summe über alle $\alpha_{ij} d_1$ beschränkt durch $d_2 - 1$, denn

$$\begin{aligned}\sum_{i=1}^{q^2} \alpha d_1 &= q^2 \alpha q^3 \\ &= q^5 \alpha \\ &< q^5 \cdot q^2 \\ &= q^7 = d_2.\end{aligned}$$

Wir bestimmen nun den Vektor $\vec{c}$ für $i \in \{1, \ldots, q\}$, $j \in \{0, 1, \ldots, r_i\}$ mit

$$c_{ij} = (i-1) d_2$$

und b mit

$$b = (q-1) d_2 + (q-1).$$

Durch eine lexikographische Sortierung der Tupel (i, j) – das heißt es gilt $(i, j) < (i', j')$, wenn $i < i'$ oder $i = i', j < j'$ gilt – können wir eine Lösung x_k von PARTIAL-UNARY-SUBSETSUM beschreiben durch

$$k \in I = \{ (i, j) \mid i \in \{1, \ldots, q\}, j \in \{1, \ldots, r_i\} \}.$$

Diese Konstruktion codiert mehre Eigenschaften, die die Wahl der Knoten und Kanten erfüllen muss, um die Instanz zu lösen. Die c_{ij} erzwingen die Wahl aller Knoten, denn nur diese erzeugen d_2. Jedoch reicht diese Wahl

nicht aus, denn jedes a_{i0} zieht d_1 in der Höhe der Knotennummer ab. Damit das zu einem Knoten i korrespondierende c_i erreicht wird, müssen daher vor dem Erreichen von c_i Kanten gewählt werden, die in i oder einem höheren Knoten münden. Somit bewegt sich die Lösung mit steigendem c_i auch im Graphen in Richtung der entsprechenden Knoten. Um zu verhindern, dass die Wahl mehrerer Kanten, die i oder einen höheren Knoten nicht erreichen, denselben Effekt erzeugen, enthält die Zielsumme b zwei weitere Eigenschaften. Zum einen besitzt sie keine Werte die mit d_1 faktorisiert sind und zum anderen erzeugt die Wahl aller Knoten insgesamt

$$-d_1 \cdot \sum_{i=2}^{q} i,$$

folglich muss die Summe der Kanten denselben Wert mit anderem Vorzeichen erzeugen. Da b weiter den nicht faktorisierten Teil $(q-1)$ enthält, muss die Summe der Kanten auch diesen erzeugen, was eine notwendige Bedingung für einen Pfad von 1 nach q ist. Um zu zeigen, dass diese Instanz genau dann lösbar ist, wenn in G ein Pfad von 1 nach q existiert, werden wir mit diesen induzierten Eigenschaften ein System konstruieren, das nur lösbar ist, wenn ein solcher Pfad existiert. Dazu zeigen wir, dass die Existenz eines solchen Pfades die Lösung der Instanz induziert, und dass diese Existenz außerdem aus einer lösbaren Instanz folgt.

Zunächst betrachten wir den Fall, dass in G ein Pfad von 1 nach q existiert. Dieser Pfad sei gegeben durch $1 = i_0 \to i_1 \to \cdots \to i_t = q$ mit $i_\mu < i_{\mu+1}$ und $\delta_\mu = i_{\mu+1} - i_\mu$. Dann wählen wir alle Knoten des Graphen und alle Kanten des Pfades, also

$$x_k = 1 \Leftrightarrow k \in \{(i,0) \mid i \in \{2,\dots,q\},\} \cup \{(i_\mu, j_\mu) \mid \mu \in \{0,\dots,t-1\}\}$$

x und sonst $x_k = 0$. Dann gilt

$$\begin{aligned}
\sum_{k\in I} a_k \cdot x_k &= \sum_{i=2}^{q} a_{i0} + \sum_{\mu=0}^{t-1} a_{i_\mu j_\mu} \\
&= \sum_{i=2}^{q} (d_2 - i d_1) + \sum_{\mu=0}^{t-1} \left(\left(i_\mu \delta_\mu + \frac{\delta_\mu(\delta_\mu+1)}{2} \right) d_1 + \delta_\mu \right) \\
&\overset{(*)}{=} (q-1)d_2 - \left(\frac{q(q+1)}{2} - 1 \right) d_1 + \left(\frac{q(q+1)}{2} - 1 \right) d_1 + (q-1) \\
&= (q-1)d_2 + (q-1) = b.
\end{aligned}$$

Denn für alle $i, \delta \in \mathbb{N}$ gilt

$$\sum_{\lambda=i+1}^{i+\delta} \lambda = \delta i + \sum_{\lambda=1}^{\delta} \lambda = \delta i + \frac{\delta(\delta+1)}{2},$$

und folglich

$$\begin{aligned}\sum_{\mu=0}^{t-1}\left(i_\mu\delta_\mu+\frac{\delta_\mu(\delta_\mu+1)}{2}\right) &= \sum_{\mu=0}^{t-1}\sum_{\lambda=i_\mu+1}^{i_\mu+\delta_\mu}\lambda\\ &= \sum_{\mu=0}^{t-1}\sum_{\lambda=i_\mu+1}^{i_{\mu+1}}\lambda\\ &= \sum_{i=2}^{q} i = \frac{q(q+1)}{2}-1. \qquad (*)\end{aligned}$$

Somit haben wir gezeigt, dass wir, wenn der Graph G einen Pfad von 1 nach q besitzt, unsere x_k so wählen können, dass die Summe über alle $a_k \cdot x_k$ die Zielsumme b ergibt. Es bleibt zu zeigen, dass für beliebige $\nu \in I$ die Summe über alle $x_k \cdot a_k$ mit $k \leqslant \nu$ größer gleich dem Wert c_ν ist. Dazu betrachten wir ein beliebiges $\nu = (i,j) \in I$ und die größte Zahl z mit $i_z < i$, sodass wir die Summe über alle Kanten (i',j') mit $i' < i$ beschreiben können.

$$\begin{aligned}\sum_{\substack{k\in I,\\ k\leqslant\nu}} a_k\cdot x_k &\geqslant \sum_{\lambda=2}^{i} a_{\lambda 0}+\sum_{\mu=0}^{z}\left(\left(i_\mu\delta_\mu+\frac{\delta_\mu(\delta_\mu)}{2}\right)d_1+\delta\mu\right)\\ &= \sum_{\lambda=2}^{i}(d_2-\lambda d_1)+d_1\sum_{\mu=0}^{z}\left(i_\mu\delta_\mu+\sum_{\lambda=1}^{\delta_\mu}\lambda\right)+\sum_{\mu=0}^{z}\delta_\mu\\ &= \sum_{\lambda=2}^{i}(d_2-\lambda d_1)+d_1\sum_{\mu=0}^{z}\sum_{\lambda=i_\mu+1}^{i_z+\delta_\mu}\lambda+\sum_{\mu=0}^{z}\delta_\mu\\ &= \sum_{\lambda=2}^{i}(d_2-\lambda d_1)+d_1\sum_{\lambda=2}^{i_{z+1}}\lambda+\sum_{\mu=0}^{z}\delta_\mu\\ &= \sum_{\lambda=2}^{i}(d_2-\lambda d_1)+d_1\sum_{\lambda=2}^{i}\lambda+\sum_{\mu=0}^{z}\delta_\mu\\ &= (i-1)d_2-\left(\frac{i(i+1)}{2}-1\right)d_1+\left(\frac{i(i+1)}{2}-1\right)d_1+(i-1)\\ &= (i-1)d_2+(i-1)\\ &\geqslant (i-1)d_2 = c_\nu.\end{aligned}$$

Folglich ist x_k mit $k \in I$ eine Lösung der Instanz.

Für die Rückrichtung sei K eine PARTIAL-UNARY-SUBSETSUM-Instanz und G der dazu korrespondierende Graph. Wir wollen zeigen, dass in G ein Weg von 1 nach q existiert, wenn K eine Lösung besitzt. Da K eine Lösung der

Form $x_k \in \{0,1\},\ k \in I$ besitzt, folgt aus

$$\sum_{\substack{(i,j)\in I,\\ (i,j)\leqslant(\nu,0)}} a_k \cdot x_k \geqslant c_{ij} = (i-1)d_2,$$

dass $x_{i0} = 1$ für alle $i \in \{2,\dots,q\}$ gilt. Das bedeutet, dass in der Lösung alle a_k, die einen Knoten repräsentieren, gewählt wurden. Wir bezeichnen nun die Anzahl der gewählten Kanten, also der $x_{ij} = 1,\ j > 0$, mit t. Diese wollen wir in sortierter Reinfolge betrachten, dazu wählen wir p_μ, j_μ für $\mu \in \{0,\dots,t-1\}$, sodass $j_\mu \neq 0, x_{p_\mu j_\mu} = 1$ und $(p_\mu, j_\mu) < (p_\nu, j_\nu)$ für $\mu < \nu$. Demnach gilt $1 \leqslant p_0 \leqslant p_1 \leqslant \cdots \leqslant p_{t-1} \leqslant q$. Weiter besitzt die Kante (p_μ, j_μ) den Abstand $\delta_{j_\mu}^{p_\mu} = \delta_\mu = j_\mu - p_\mu$ und wir erhalten die folgenden drei Aussagen:

(i) Zunächst zeigen wir, dass eine vom Startknoten ausgehende Kante gewählt wurde. Nehmen wir an, dies wäre nicht der Fall, also $x_{1j} = 0$ für alle j, dann

$$\sum_{\substack{k\in I,\\ k\leqslant(2,0)}} a_k \cdot x_k = a_{20} = d_2 - 2d_1 < d_2 = c_{20}.$$

Dies ist ein Widerspruch, denn K besitzt eine Lösung. Folglich gilt $p_0 = 1$ und der erste Knoten des durch die Lösung induzierten Pfades ist der Startknoten.

(ii) Da K eine Lösung besitzt, gilt

$$\sum_{k\in I} a_k \cdot x_k = b.$$

Wir haben bereits gezeigt, dass in der Lösung alle Knoten gewählt wurden, weiter wurden t Kanten gewählt. Äquivalent zur Hinrichtung formulieren wir

$$\sum_{k\in I} a_k \cdot x_k = \underbrace{(q-1)d_2 - d_1 \sum_{i=2}^{q} i}_{\text{gewählten Knoten}} + \underbrace{\sum_{\mu=0}^{t-1} \left(\left(p_\mu \delta_\mu + \frac{\delta_\mu(\delta_\mu+1)}{2} \right) d_1 + \delta_\mu \right)}_{\text{gewählten Kanten}}$$
$$= b = (q-1)d_2 + q - 1.$$

Dies können wir umformen und erhalten die Strecke der gewählten Kanten

$$\sum_{\mu=0}^{t-1} \delta_\mu = q - 1,$$

sowie

$$\sum_{\mu=0}^{t-1} \left(p_\mu \delta_\mu + \frac{\delta_\mu(\delta_{\mu+1})}{2} \right) = \sum_{i=2}^{q} i.$$

Dass die Strecke der gewählten Kanten $q-1$ beträgt, ist eine notwendige Bedienung für einen Pfad von 1 nach q. Dies, sowie die zweite Gleichung, werden wir im zweiten Teil der Rückrichtung benötigen.

(iii) Analog zu (ii) wollen wir nun die Eigenschaft

$$\sum_{\substack{k\in I,\\ k\leqslant(p_\nu,0)}} a_k \cdot x_k \geqslant c_{p_\nu,0}$$

ausnutzen. Diese Ungleichung lässt sich mit der gleichen Rechnung formulieren als

$$\sum_{\substack{k\in I,\\ k\leqslant(p_\nu,0)}} a_k \cdot x_k = \underbrace{(p_\nu-1)d_2 - d_1 \sum_{i=2}^{p_\nu} i}_{\text{gewählte Knoten}} + \underbrace{\sum_{\mu=0}^{\nu-1}\left(\left(p_\mu\delta_\mu + \frac{\delta_\mu(\delta_\mu+1)}{2}\right)d_1 + \delta_\mu\right)}_{\text{gewählte Kanten}}$$
$$\geqslant c_{(p_\nu,0)} = (p_\nu-1)d_2.$$

Demnach muss für alle $\nu \in \{0,\dots,t-1\}$ Folgendes gelten:

$$\sum_{\mu=0}^{\nu-1}\left(p_\mu\delta_\mu + \frac{\delta_\mu(\delta_\mu+1)}{2}\right) \geqslant \sum_{i=2}^{p_\nu} i.$$

Dabei müssen in der zweiten Ungleichung die δ_μ nicht beachtet werden, da die Summe aller δ_μ, wie gezeigt, durch d_1-1 beschränkt ist.

Jetzt können wir mit Lemma 18 zeigen, dass die gewählten Kanten in G einen Pfad von 1 nach q bilden. Dazu wollen wir zunächst ein System konstruieren, welches Lemma 18 genügt. Wir setzen $p_t = q$, dies ist möglich – denn die Werte p_i mit $i < t$ beschreiben stets nur den Ausgangsknoten einer Kante und aufgrund der topologischen Sortierung kann diese Belegung noch nicht existieren. Wir führen nun den Sollabstand $\Delta_i = p_i - p_{i-1}$ ein, den eine Kante besitzen muss, wenn die p_i einen Pfad von 1 nach q induzieren. Weiter sei für jede Kante der Wert $y_i = \delta_{i-1} - \Delta_i$. Demnach müssen wir $\vec{y} = \vec{0}$ zeigen und konstruieren dazu im Folgenden ein entsprechendes System.

(i') Durch die topologische Ordnung der Knoten sind die Abstände aller Kanten positiv, also $\delta_{i-1} > 0$, und folglich gilt $y_i > -\Delta_i = p_{i-1} - p_i$.

Wegen (i) und (ii) folgt weiter

$$\begin{aligned}
\sum_{i=1}^{t} y_i &= \sum_{i=1}^{t} (\delta_{i-1} - \Delta_i) \\
&= \sum_{i=1}^{t} \delta_{i-1} - \sum_{i=1}^{t} \Delta_i \\
&= \sum_{i=0}^{t-1} \delta_i - \sum_{i=1}^{t} \Delta_i \\
&\overset{(ii)}{=} (q-1) - \sum_{i=1}^{t} \Delta_i \\
&= (q-1) - \big((p_1 - p_0) + (p_2 - p_1) + \cdots + (p_t - p_{t-1})\big) \\
&= (q-1) - (p_t - p_0) \\
&\overset{(i)}{=} 0.
\end{aligned}$$

(ii′) Für unser System müssen wir nun $\sum_{i=1}^{t} \left(p_i y_i + \frac{y_i(y_i+1)}{2}\right)$ berechnen. Wir erhalten $p_i y_i = (p_{i-1} + \Delta_i) y_i$ durch $\Delta_i = p_i - p_{i-1}$. Des Weiteren gilt durch Erweiterung

$$\frac{y_i(y_i+1)}{2} = \frac{(y_i+\Delta_i)(y_i+\Delta_i+1)}{2} - \frac{\Delta_i(\Delta_i+1)}{2} - \Delta_i y_i.$$

Außerdem gilt

$$(p_{i-1} + \Delta_i) y_i - \Delta_i y_i = p_{i-1} y_i = (y_i + \Delta_i) p_{i-1} - \Delta_i p_{i-1}$$

und mit $y_i + \Delta_i = \delta_{i-1}$ folgt noch

$$\frac{(y_i+\Delta_i)(y_i+\Delta_i+1)}{2} = \frac{\delta_{i-1}(\delta_{i-1}+1)}{2}.$$

Zusammen erhalten wir

$$\begin{aligned}
p_i y_i + \frac{y_i(y_i+1)}{2} &= (p_{i-1} + \Delta_i) y_i \\
&\quad + \frac{(y_i+\Delta_i)(y_i+\Delta_i+1)}{2} - \frac{\Delta_i(\Delta_i+1)}{2} - \Delta_i y_i \\
&= p_{i-1}(y_i + \Delta_i) \\
&\quad - \Delta_i p_{i-1} + \frac{\delta_{i-1}(\delta_{i-1}+1)}{2} - \frac{\Delta_i(\Delta_i+1)}{2}.
\end{aligned}$$

Weiter gilt

$$\begin{aligned}
-\Delta_i p_{i-1} - \frac{\Delta_i(\Delta_i+1)}{2} &= -\Delta_i p_{i-1} - \sum_{\nu=1}^{\Delta_i} \nu \\
&= -(p_i - p_{i-1})p_{i-1} - \sum_{\nu=1}^{(p_i-p_{i-1})} \nu \\
&= -\sum_{\nu=p_{i-1}+1}^{(p_i-p_{i-1})+p_{i-1}} \nu \\
&= -\sum_{\nu=p_{i-1}+1}^{p_i} \nu.
\end{aligned}$$

Damit können wir nun den zu Anfang beschriebenen Ausdruck berechnen zu

$$\begin{aligned}
\sum_{i=1}^{t}\left(p_i y_i + \frac{y_i(y_i+1)}{2}\right) &= \sum_{i=0}^{t-1}\left(p_{i+1}y_{i+1} + \frac{y_{i+1}(y_{i+1}+1)}{2}\right) \\
&= \sum_{i=0}^{t-1}\Bigg(p_i(y_{i+1}+\Delta_{i+1}) - \Delta_{i+1}p_i \\
&\qquad + \frac{\delta_i(\delta_i+1)}{2} - \frac{\Delta_{i+1}(\Delta_{i+1}+1)}{2}\Bigg) \\
&= \sum_{i=0}^{t-1}\left(p_i(y_{i+1}+\Delta_{i+1}) + \frac{\delta_i(\delta_i+1)}{2}\right) \\
&\qquad - \sum_{i=0}^{t-1}\left(\Delta_{i+1}p_i + \frac{\Delta_{i+1}(\Delta_{i+1}+1)}{2}\right) \\
&= \sum_{i=0}^{t-1}\left(p_i(y_{i+1}+\Delta_{i+1}) + \frac{\delta_i(\delta_i+1)}{2}\right) \\
&\qquad - \sum_{i=1}^{t}\left(\Delta_i p_{i-1} + \frac{\Delta_i(\Delta_i+1)}{2}\right) \\
&= \sum_{i=0}^{t-1}\left(p_i\delta_i + \frac{\delta_i(\delta_i+1)}{2}\right) - \sum_{i=1}^{t}\sum_{\nu=p_{i-1}+1}^{p_i}\nu \\
&\overset{(ii)}{=} \sum_{i=2}^{q} i - \sum_{\nu=p_0+1}^{p_t}\nu \\
&\overset{(i)}{=} \sum_{i=2}^{q} i - \sum_{i=2}^{q} i = 0.
\end{aligned}$$

(iii′) Analog zu den Berechnungen in (ii′) können wir den letzten Teil unseres Systems wie folgt berechnen

$$\begin{aligned}\sum_{i=1}^{\nu}\left(p_i y_i + \frac{y_i(y_i+1)}{2}\right) &= \sum_{i=0}^{\nu-1}\left(p_i\delta_i + \frac{\delta_i(\delta_i+1)}{2}\right) - \sum_{i=1}^{\nu}\sum_{\mu=p_{i-1}+1}^{p_i}\mu \\ &\overset{(iii)}{\geqslant} \sum_{i=2}^{p_\nu} i - \sum_{i=1}^{\nu}\sum_{\mu=p_{i-1}+1}^{p_i}\mu \\ &= \sum_{i=2}^{p_\nu} i - \sum_{\mu=p_0+1}^{p_\nu}\mu \\ &= 0.\end{aligned}$$

In (i′), (ii′) und (iii′) haben wir gezeigt, dass $y_1, \ldots, y_t$ das System

$$\begin{aligned}\sum_{i=1}^{t} y_i &= 0 \\ \sum_{i=1}^{t}\left(p_i y_i + \frac{y_i(y_i+1)}{2}\right) &= 0 \\ \sum_{i=1}^{\nu}\left(p_i y_i + \frac{y_i(y_i+1)}{2}\right) &\geqslant 0 \\ y_i &> p_{i-1} - p_i\end{aligned}$$

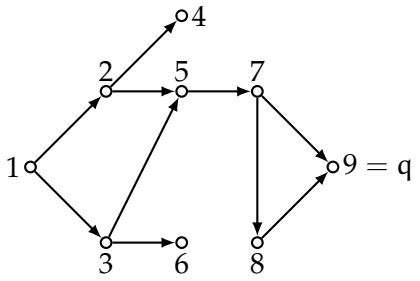
löst. Nach Lemma 18 hat dieses System nur die Lösung $\vec{y} = \vec{0}$. Folglich muss $\delta_{i-1} = \Delta_i$ für alle $i \in \{1, \ldots, t-1\}$ gelten und das bedeutet, dass $1 = p_0 \to p_1 \to p_2 \to \cdots \to p_t = q$ ein Pfad im Graphen G ist.

Wir haben gezeigt, dass wir zu jeder Instanz von DAG-REACH eine entsprechende Instanz von PARTIAL-UNARY-SUBSETSUM konstruieren können, die genau dann lösbar ist, wenn die Ausgangsinstanz lösbar ist. Es bleibt zu zeigen, dass wir für eine gegebene Instanz von DAG-REACH eine korrespondierende Instanz von PARTIAL-UNARY-SUBSETSUM auf logarithmischem Platz berechnen können. Dies ist möglich, denn eine deterministische Turingmaschine kann die Werte a_{ij}, c_{ij} und b mit logarithmischen Platzverbrauch berechnen und ausgeben. Wir haben DAG-REACH auf PARTIAL-UNARY-SUBSETSUM reduziert und folglich ist PARTIAL-UNARY-SUBSETSUM ein NL-hartes Problem. □

► Korollar 20
Das Problem PARTIAL-UNARY-SUBSETSUM ist NL-vollständig. ◁

▶ Beispiel 21

Das folgende Beispiel soll die Reduktion von einer DAG-REACH-Instanz auf PARTIAL-UNARY-SUBSETSUM veranschaulichen. Dazu betrachten wir einen Graphen $G = (V, E^G)$ mit den Knoten $V = \{1, 2, \ldots, 9\}$ und der in der folgenden Abbildung dargestellten Kantenrelation E^G.



In G sind 1 und $9 = q$ der Start- bzw. Zielknoten der DAG-REACH-Instanz, weiter bezeichnen wir die Nachbarn von jedem Knoten i mit $\{1, \ldots, r_i\}$. Wir berechnen nun den Vektor $\vec{a}$ einer PARTIAL-UNARY-SUBSETSUM-Instanz, indem wir für jeden Knoten $1, i \in V$ mit $i > 1$ die Werte

$$\begin{aligned} a_{10} &= 0, \\ a_{i0} &= d_2 - i \cdot d_1, \ i > 1 \end{aligned}$$

und für jede Kante $(i, j) \in E$ mit $i < j$ und Abstand $\delta^i_j = j - i$ den Wert

$$a_{ij} = \left(i \cdot \delta^i_j + \frac{\delta^i_j(\delta^i_j + 1)}{2} \right) \cdot d_1 + \delta^i_j$$

bestimmen. Den Vektor $\vec{c}$ definieren wir dann durch $c_{ij} = (i-1)d_2$ für $i \in \{1, \ldots, q\}$ und $j \in \{0, 1, \ldots, r_i\}$. Die Zielsumme setzen wir auf

$$b = (q-1)d_2 + (q-1) = 8d_2 + 8.$$

Weiter sei

$$I = \{(i,j) \mid i \in \{1,\dots,q\}, j \in \{1,\dots,r_i\}\}$$

das Intervall, über das wir später iterieren. Die a_{ij} und c_{ij} berechnen sich somit wie in der Tabelle rechts zu sehen, dabei entspricht der Wert j der Nummer des Nachbarn und der Wert in Klammern dem Knotenwert im Graphen. Der Graph G als Instanz von DAG-REACH besitzt die Lösung

$$1 \to 2 \to 5 \to 7 \to 9 = q$$

der Form

$$i_0 \to i_1 \to \cdots \to i_t = q$$

mit $i_\mu < i_{\mu+1}$ für alle $\mu \in \{0,\dots,t-1\}$. Wir setzen $\delta_\mu = i_{\mu+1} - i_\mu$, dann existiert zu jedem μ ein $j_\mu \in \{1,\dots,r_{i_\mu}\}$ mit $\delta_\mu = \delta_{j_\mu}^{i_\mu}$. Das bedeutet, dass wir für jede Kante unseres Pfades eine Kante mit entsprechendem δ finden. Über diese Kanten iterieren wir mit μ.

i	j	a_{ij}	c_{ij}
1	0	0	0
	1(2)	$2d_1+1$	0
	2(3)	$5d_1+2$	0
2	0	d_2-2d_1	d_2
	1(4)	$7d_1+2$	d_2
	2(5)	$12d_1+3$	d_2
3	0	d_2-3d_1	$2d_2$
	1(5)	$9d_1+2$	$2d_2$
	2(6)	$15d_1+3$	$2d_2$
4	0	d_2-4d_1	$3d_2$
5	0	d_2-5d_1	$4d_2$
	1(7)	$13d_1+2$	$4d_2$
6	0	d_2-6d_1	$5d_2$
7	0	d_2-7d_1	$6d_2$
	1(8)	$8d_1+1$	$6d_2$
	2(9)	$17d_1+2$	$6d_2$
8	0	d_2-8d_1	$7d_2$
	1(9)	$9d_1+1$	$7d_2$
9	0	d_2-9d_1	$8d_2$

Nun wählen wir $x_k = 1$, wenn

$$k \in \{(i,0) \mid i \in \{2,\dots,q\}\} \cup \{(i_\mu, j_\mu) \mid \mu \in \{0,\dots,t-1\}\}$$

gilt und sonst $x_k = 0$. Es folgt

$$\begin{aligned}
\sum_{k\in I} a_k x_k &= \sum_{i=2}^{q} a_{i0} + \sum_{\mu=0}^{t-1} a_{i_\mu j_\mu} \\
&= (8d_2 - 44d_1) + \sum_{\mu=0}^{t-1} a_{i_\mu j_\mu} \\
&= (8d_2 - 44d_1) + (a_{11} + a_{22} + a_{51} + a_{72}) \\
&= (8d_2 - 44d_1) + (44d_1 + 8) \\
&= (8d_2 + 8) \\
&= b.
\end{aligned}$$

Aus der folgenden Tabelle geht hervor, dass wir für alle $\nu \in I$ die Bedingungen für c_ν erfüllen.

i	j	$\sum_{k\in I, k\leqslant(i,j)}$	c_{ij}
1	1	$2d_1+1$	0
2	0	d_2+1	d_2
2	2	d_2+12d_1+4	d_2
3	0	$2d_2+9d_1+4$	$2d_2$
4	0	$3d_1+5d_1+4$	$3d_2$
5	0	$4d_2+4$	$4d_2$
5	1	$4d_2+13d_1+6$	$4d_2$
6	0	$5d_2+7d_1+6$	$5d_2$
7	0	$6d_2+6$	$6d_2$
7	2	$6d_2+17d_1+8$	$6d_2$
8	0	$7d_2+9d_1+8$	$7d_2$
9	0	$7d_2+8$	$8d_2$

$\triangleleft$

4 Neue Komplexitätsaussagen zu weiteren Varianten von UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM

In diesem Abschnitt betrachten wir die Berechnungskomplexität von weiteren Varianten von UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM. Wir werden zunächst einige Varianten mit mehren Dimensionen, wie zum Beispiel das bekannte UNARY-KNAPSACK-Problem, betrachten. Im zweiten Teil dieses Abschnittes betrachten wir dann Varianten mit einer anderen Beschränkung – hier werden wir beispielsweise die untere Schranke des Problems PARTIAL-UNARY-SUBSETSUM um eine zusätzliche obere Schranke ergänzen. Im letzten Teil diskutieren wir Varianten von UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM, die über einem anderen Grundkörper, wie zum Beispiel den ganzen Zahlen, definiert sind.

4.1 Verschiedene mehrdimensionale Versionen von UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM

Wir beschäftigen uns zunächst mit Varianten von UNARY-SUBSETSUM, die das Ausgangsproblem auf mehrere Dimensionen verallgemeinern. Die erste Variante ist das in der Informatik sehr bekannte KNAPSACK-Problem in unärer Kodierung. Informell lässt sich dieses Problem wie folgt beschreiben: Ein Dieb bricht in ein Haus ein und möchte Waren mit einem möglichst hohen Wert stehlen. Da der Dieb zu Fuß unterwegs ist und nur einen Rucksack dabei hat, ist die gesamt Kapazität, die er stehlen kann, beschränkt. Die Frage ist nun, welche Gegenstände er einpacken soll, damit er einen möglichst hohen Wert erzielt und dennoch alle gewählten Gegenstände in seinen Rucksack passen. Beschreibt g den gewünschten Zielwert der Waren und w die maximale Kapazität des Rucksacks, so lässt sich das Problem formal wie folgt beschreiben:

▶ Problem 22 (Knapsack)

Eingabe: Die Vektoren $\vec{a}, \vec{b} \in \mathbb{N}^p$ mit $p \in \mathbb{N}$ sowie eine Zielsumme $g \in \mathbb{N}$ und ein Maximalgewicht $w \in \mathbb{N}$.

Gesucht: Ein Vektor $\vec{x} \in \{0,1\}^p$, sodass Folgendes gilt:

$$\sum_{i=1}^{p} x_i \cdot a_i = g \quad \text{und} \quad \sum_{i=1}^{p} x_i \cdot b_i \leqslant w.$$

Sprache: $\textsc{unary-knapsack} = \{ 0^{a_1}10^{b_1}11 \ldots 110^{a_p}10^{b_p}1110^{g}10^{w} \mid \exists I \subseteq \{1,\ldots,p\} \centerdot \sum_{i\in I} a_i = g \wedge \sum_{i\in I} b_i \leqslant w \}$. ◁

Ist der Dieb ein Meister seines Handwerkes, so ist sein Leben nicht so leicht wie oben beschrieben. Für einen wirklich erfolgreichen Raubzug müssen viele Kleinigkeiten beachtet werden. Zum einen ist ein Raubzug gefährlich, denn jederzeit könnte der Alarm auslösen oder der Besitzer des Hauses überraschend zurückkehren. In diesem Fall müsste der Dieb sofort fliehen, ohne den Rucksack weiter packen zu können. Es ist daher wichtig, dass der erfahrene Dieb zuerst die Gegenstände mit hohem Wert einpackt, damit er im Falle der Flucht noch eine möglichst gute Ausbeute erzielt. Des Weiteren muss der Dieb beim Packen des Rucksacks die schweren Gegenstände nach unten legen, denn es wäre zu schade, wenn eine schwere Goldstatue eine zerbrechliche Glaskugel zerdrückt und somit wertlos macht. Dieses Szenario des Meisterdiebes wird formal durch folgendes Problem beschrieben, wenn g und w wie oben definiert sind und $\vec{c}$ sicherstellt, dass zunächst Gegenstände mit einem hohen Wert gewählt werden, und weiter $\vec{d}$ dafür sorgt, dass zuerst schwere Gegenstände eingepackt werden:

▶ Problem 23 (Partial-Rucksack)

Eingabe: Die Vektoren $\vec{a}, \vec{b}, \vec{c}, \vec{d} \in \mathbb{N}^p$ mit $p \in \mathbb{N}$ sowie eine Zielsumme $g \in \mathbb{N}$ und ein Maximalgewicht $w \in \mathbb{N}$.

Gesucht: Ein Vektor $\vec{x} \in \{0,1\}^p$, sodass die folgende Gleichungen erfüllt sind:

$$\sum_{i=1}^{p} x_i \cdot a_i = g, \quad \sum_{i=1}^{p} x_i \cdot b_i \leqslant w,$$
$$\sum_{i=1}^{\nu} x_i \cdot a_i \geqslant c_\nu \quad \text{für alle } \nu \in \{1,\ldots,p\},$$
$$\sum_{i=1}^{\nu} x_i \cdot b_i \geqslant d_\nu \quad \text{für alle } \nu \in \{1,\ldots,p\}.$$

Sprache: $\textsc{partial-unary-knapsack} = \{ 0^{a_1}10^{b_1}10^{c_1}10^{d_1}11 \ldots 110^{a_p}10^{b_p}10^{c_p}10^{d_p}1110^{g}10^{w} \mid \exists I \subseteq \{1,\ldots,p\} \centerdot \sum_{i\in I} a_i = g \wedge \sum_{i\in I} b_i \leqslant w \wedge \forall_{\nu\leqslant p} \centerdot \sum_{i\in I, i\leqslant\nu} a_i \geqslant c_\nu \wedge \sum_{i\in I, i\leqslant\nu} b_i \geqslant d_\nu \}$. ◁

Zunächst betrachten wir die Komplexität von UNARY-KNAPSACK, anschließend untersuchen wir jene von PARTIAL-UNARY-KNAPSACK.

► Satz 24
Das Problem UNARY-KNAPSACK ist in unärer Codierung TC^0-vollständig.

Beweis: Zunächst zeigen wir, dass UNARY-KNAPSACK $\in TC^0$ gilt. Dazu verwenden wir Satz 12 und betrachten eine Instanz von UNARY-SUBSETSUM als Graphstruktur mit beschränkter Baumtiefe. Wir modellieren die Eingabe als Graphen $G = (V, E^G)$ mit einer Färbung der Knoten, sodass V in zwei disjunkte Mengen $V_{\oslash}$ und $V_{\square}$ zerfällt. Für jedes $i \in \{1, \dots, p\}$ konstruieren wir einen $\oslash$-Stern mit a_i Knoten und einen $\square$-Stern mit b_i Knoten. Die Zentralknoten der zu a_i und b_i korrespondierenden Sterne verbinden wir durch eine Kante. Wir beschreiben diesen Graphen über der Signatur $\tau = (E^2, C^1_{\oslash}, C^1_{\square})$ mit der Struktur $G = (V, E^G, C^G_{\oslash}, C^G_{\square})$, wobei V und E^G wie üblich definiert sind und die Relationen $C^G_{\oslash}$ und $C^G_{\square}$ eine $\oslash$- bzw. $\square$-Färbung eines Knoten beschreiben. Die Konstruktion aus zwei solchen Sternen besitzt die konstante Baumtiefe 3, demnach ist die Baumtiefe des gesamten Graphen nach oben beschränkt. Wir betrachten nun die MSO-Formel $\varphi(X, Y)$ mit

$$\begin{aligned}\varphi(X, Y) = \forall x \,.\, \big(X(x) \to C_{\oslash}(x)\big) \wedge \big(Y(x) \to C_{\square}(x)\big) \\ \wedge \,.\, X(x) \vee Y(x) \to \forall y \,.\, E(x, y) \to X(y) \vee Y(y).\end{aligned}$$

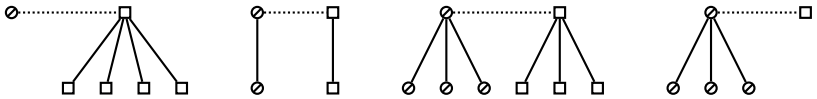
Diese Formel sagt aus, dass die Menge X nur $\oslash$-Sterne und die Menge Y nur $\square$-Sterne vollständig enthalten kann. Dabei meint vollständig, dass entweder alle Knoten des Sterns in der Menge liegen oder keiner. Da die zu a_i und b_i korrespondierenden Sterne verbunden sind folgt weiter, dass Y den Stern zu b_i enthält, wenn der Stern zu a_i in X liegt. Da die Baumtiefe des Graphen beschränkt ist, können wir nach Satz 12 die Bits des Lösungshistogramm $\text{histogram}(G, \varphi)$ mit TC^0-Schaltkreisen berechnen. Betrachten wir das Histogramm als Tabelle, so beschreibt Zeile i die Menge der Lösungen, wenn die Menge X genau i Knoten enthält, und Spalte j beschreibt die Anzahl der Lösungen, wenn Y genau j Knoten enthält. Wir müssen nun prüfen, ob ein Eintrag $\text{histogram}(G, \varphi)[g][\nu] > 0$ für ein beliebiges $\nu \leqslant w$ existiert. Dazu berechnen TC^0-Schaltkreise die entsprechenden Bits im Lösungshistogramm und ein weiterer TC^0-Schaltkreis prüft mit einem Threshold-Gatter, ob mindestens eines dieser Bits eine 1 ist. Die Anzahl der benötigten TC^0-Schaltkreise und somit die Anzahl der verwendeten Gatter ist polynomiell in der Eingabelänge beschränkt, da die Eingabe unär und der Zähler im Histogramm binär codiert ist. Ist dies der Fall, so können wir Sterne so wählen, dass wir genau g Knoten von $\oslash$-Sternen wählen, ohne mehr als w Knoten von $\square$-Sternen zu erhalten und folglich ist die Instanz von UNARY-KNAPSACK dann lösbar. Es folgt UNARY-KNAPSACK $\in TC^0$.

Um die TC^0-Härte von UNARY-KNAPSACK zu zeigen, wollen wir von MAJORITY

reduzieren. Dies gelingt mit einer FO(BIT)-Reduktion, wie wir sie in Satz 13 bereits ausführlich besprochen haben. Sei m der Binärstring der Länge p einer MAJORITY-Instanz und seien weiter $\vec{a}$, $\vec{b}$ die Bezeichner für die Eingabevektoren einer UNARY-KNAPSACK-Instanz, dann setzen wir $a_i = b_i = m_i$ für $i \in \{1, \ldots, p\}$ und $g = w = \lfloor p/2 \rfloor$. Ist die Instanz von MAJORITY lösbar, so ist das Gewicht von $m \geqslant \lfloor p/2 \rfloor$, folglich lassen sich in UNARY-KNAPSACK Werte so aufaddieren, dass sie g ergeben. Da weiter $\vec{a} = \vec{b}$ gilt, wird w in der Summe zwar erreicht, jedoch nicht überschritten. Entsprechend ist auch die Instanz von UNARY-KNAPSACK lösbar. Ist wiederum MAJORITY nicht lösbar, so ist das Gewicht von m kleiner als $\lfloor p/2 \rfloor$ – entsprechend können in $\vec{a}$ keine Werte so aufaddiert werden, dass sie in der Summe g bilden und folglich ist die Instanz von UNARY-KNAPSACK nicht lösbar. Es folgt die TC^0-Härte von UNARY-KNAPSACK. □

► Beispiel 25

Wir betrachten eine Instanz von UNARY-KNAPSACK, gegeben durch die Vektoren $\vec{a} = (1, 2, 3, 4)$, $\vec{b} = (5, 2, 3, 1)$ sowie $g = 5$ und $w = 10$. Der zu dieser UNARY-KNAPSACK-Instanz korrespondierende Graph aus Sternen ist in der folgenden Abbildung dargestellt. Dabei stehen ⊘-Knoten für Werte und □-Knoten für Gewichte. Die Kante zwischen zwei Sterne a_i und b_i ist gepunktet dargestellt.



Das Lösungshistogramm histogram(G, φ) der UNARY-KNAPSACK-Instanz ist in der Tabelle auf der nächsten Seite abgebildet. Die Zeilen beschreiben die Kardinalität von X, also die Anzahl der gewählten Werteknoten. Die Spalten beschreiben die Kardinalität von Y, also die Anzahl der gewählten Gewichtsknoten. Ist ein Eintrag (x, y) größer als 0, so lässt sich ein Wert von x unter Verwendung von y Gewicht erreichen. Zeile 5 (hervorgehoben) zeigt, dass wir den Zielwert $g = 5$ erreichen können, ohne das Maximalgewicht $w = 10$ zu überschreiten, denn Zeile 5 ist in Spalte 6 und 9 größer als 0.

$\lvert X\rvert, \lvert Y\rvert$	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	1	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	1	0	0	0	0	0
4	1	0	0	1	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	1	0	0	0
6	0	0	1	0	0	1	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	1	0
8	0	0	0	0	1	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1	0	0
10	0	0	0	0	0	0	1	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	1

◁

Um die Komplexität von PARTIAL-UNARY-KNAPSACK zu untersuchen, wollen wir zunächst einen NL-Algorithmus angeben und anschließend vom Problem PARTIAL-UNARY-SUBSETSUM reduzieren.

► Satz 26
Das Problem PARTIAL-UNARY-KNAPSACK ist NL-vollständig.

Beweis: Wir betrachten eine Instanz von PARTIAL-UNARY-KNAPSACK gegeben durch

$$0^{a_1}10^{b_1}10^{c_1}10^{d_1}110^{a_2}10^{b_2}10^{c_2}10^{d_2}11\ldots0^{a_p}10^{b_p}10^{c_p}10^{d_p}1110^{g}10^{w}.$$

Eine nichtdeterministische Turingmaschine mit zwei Arbeitsbändern kann diese Eingabe von links nach rechts lesen und dabei einige $i \in \{1,\ldots,p\}$ raten, für welche sie a_i auf ihrem ersten Arbeitsband und b_i auf ihrem zweiten Arbeitsband aufsummiert. Wann immer sie für $\nu \in \{1,\ldots,p\}$ ein c_ν erreicht, prüft sie, ob die Summe auf dem ersten Band größer gleich diesem c_ν ist und verwirft, wenn dies nicht der Fall ist. Einen entsprechenden Vergleich führt sie auch für jedes d_ν mit dem Wert auf dem zweiten Band durch. Beim Erreichen des Endes der Eingabe prüft sie, ob der Wert auf dem ersten Band gleich g ist und ob die Summe auf dem zweiten Band kleiner gleich w ist. Wenn beides der Fall ist, akzeptiert die Turingmaschine. Da die Eingabe unär codiert ist, können die Berechnungen auf beiden Bändern – unter Verwendung von im Bezug zur Eingabelänge logarithmischem Platz – durchgeführt werden und folglich gilt: PARTIAL-UNARY-KNAPSACK $\in$ NL.

Aus Korollar 20 wissen wir, dass PARTIAL-UNARY-SUBSETSUM NL-vollständig ist. Dies wollen wir nun nutzen und von PARTIAL-UNARY-SUBSETSUM auf PARTIAL-UNARY-KNAPSACK reduzieren. Dazu betrachten wir eine Instanz von PARTIAL-UNARY-SUBSETSUM gegeben durch den Eingabevektor $\vec{A} \in \mathbb{N}^p$, den Gewichtsvektor $\vec{C} \in \mathbb{N}^p$, der Zielsumme $B \in \mathbb{N}$ und $p \in \mathbb{N}$. Wir wollen daraus eine wie oben eingeführte Instanz der Form $\vec{a}, \vec{b}, \vec{c}, \vec{d} \in \mathbb{N}^p$ mit

$g, w \in \mathbb{N}$ für das Problem PARTIAL-UNARY-KNAPSACK konstruieren. Wir setzen dazu $g = w = B, \vec{a} = \vec{b} = \vec{A}$ und $\vec{c} = \vec{d} = \vec{C}$. Es bleibt zu zeigen, dass diese Instanz von PARTIAL-UNARY-KNAPSACK genau dann lösbar ist, wenn auch jene von PARTIAL-UNARY-SUBSETSUM lösbar ist. Ist die Instanz von PARTIAL-UNARY-SUBSETSUM lösbar, dann gilt:

$$\sum_{i=1}^{p} x_i \cdot A_i = B,$$
$$\sum_{i=1}^{\nu} x_i \cdot A_i \geqslant C_\nu \quad \text{für alle } \nu \in \{1, \ldots, p\}.$$

Dann gilt in der konstruierten PARTIAL-UNARY-KNAPSACK-Instanz Folgendes:

$$\sum_{i=1}^{p} x_i \cdot a_i = g, \qquad \sum_{i=1}^{\nu} x_i \cdot a_i \geqslant c_\nu \quad \text{für alle } \nu \in \{1, \ldots, p\},$$
$$\sum_{i=1}^{p} x_i \cdot b_i = w, \qquad \sum_{i=1}^{\nu} x_i \cdot b_i \geqslant d_\nu \quad \text{für alle } \nu \in \{1, \ldots, p\}.$$

Folglich ist auch die Instanz von PARTIAL-UNARY-KNAPSACK lösbar. Nehmen wir nun an, die Instanz von UNARY-SUBSETSUM wäre nicht lösbar. Dann ist entweder die Summe insgesamt ungleich B oder eine Teilsumme ist kleiner als ein C_ν. Entsprechend würde für PARTIAL-UNARY-KNAPSACK gelten, dass die Summe insgesamt ungleich g ist oder aber, dass eine Teilsumme ein c_ν nicht erreicht. Demnach wäre die Instanz von PARTIAL-UNARY-KNAPSACK nicht lösbar. Da nur Elemente der UNARY-SUBSETSUM-Instanz kopiert werden müssen, kann die Reduktion von einer Turingmaschine mit logarithmischer Platzschranke durchgeführt werden und es folgt die Aussage. □

Die oben beschriebenen UNARY-KNAPSACK-Varianten verallgemeinern das Problem UNARY-SUBSETSUM auf eine zweidimensionale Eingabe. Wir verallgemeinern die Eingabe im Folgenden weiter und betrachten Instanzen von UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM, die eine Eingabe einer beliebigen Dimension erwarten.

▶ Problem 27 (ℓ-Subset-Sum)

Eingabe: Vektoren $\vec{a_1}, \ldots, \vec{a_\ell} \in \mathbb{N}^p$ mit $p \in \mathbb{N}$ sowie ein Zielvektor $\vec{s} \in \mathbb{N}^\ell$ mit $\ell \in \mathbb{N}$.

Gesucht: Ein Vektor $\vec{x} \in \{0,1\}^p$, sodass folgendes gilt:

$$\sum_{i=1}^{p} x_i \cdot (\vec{a_k})_i = s_k \quad \text{für alle } k \in \{1, \ldots, \ell\}.$$

Dabei beschreibt die doppelte Indizierung $(\vec{a_k})_i$ das i*te* Element des Vektors $\vec{a_k}$.

Sprache: ℓ-UNARY-SUBSETSUM $= \{\, \vec{a_1}, \ldots, \vec{a_\ell}, \vec{s}$ in unärer Codierung $|$
es existiert ein $\vec{x}$, sodass die obigen Gleichungen erfüllt sind $\}$. ◁

▶ Problem 28 (ℓ-Partial Subset-Sum)

Eingabe: Vektoren $\vec{a_1}, \ldots, \vec{a_\ell}, \vec{c_1}, \ldots, \vec{c_\ell}, \in \mathbb{N}^p$ mit $p \in \mathbb{N}$ sowie ein Zielvektor $\vec{s} \in \mathbb{N}^\ell$ mit $\ell \in \mathbb{N}$.

Gesucht: Ein Vektor $\vec{x} \in \{0,1\}^p$, der für alle $k \in \{1, \ldots, \ell\}$ die folgenden Gleichungen erfüllt:

$$\sum_{i=1}^{p} x_i \cdot (\vec{a_k})_i = s_k,$$
$$\sum_{i=1}^{\nu} x_i \cdot (\vec{a_k})_i \geqslant (\vec{c_k})_\nu \quad \text{für alle } \nu \in \{1, \ldots, p\}.$$

Sprache: ℓ-PARTIAL-UNARY-SUBSETSUM $= \{$
$\vec{a_1}, \ldots, \vec{a_\ell}, \vec{c_1}, \ldots, \vec{c_\ell}, \vec{s}$ in unärer Codierung $|$
es existiert ein $\vec{x}$, sodass die obigen Gleichungen erfüllt sind $\}$. ◁

▶ Satz 29

Das Problem ℓ-UNARY-SUBSETSUM ist TC^0-vollständig.

Beweis: Wir zeigen zunächst mit Satz 12, dass ℓ-UNARY-SUBSETSUM $\in TC^0$ gilt. Dazu konstruieren wir zu einer Instanz von ℓ-UNARY-SUBSETSUM eine Graphenstruktur G, indem wir für jedes $i \in \{1, \ldots, p\}$ und jedes $k \in \{1, \ldots, \ell\}$ einen Stern mit $(\vec{a_k})_i$ Knoten erstellen. Wir verbinden die Zentralknoten alle Sterne, die einen Wert $(\vec{a_k})_j$ für $k \in \{1, \ldots, \ell\}$ repräsentieren, miteinander und färben alle Sterne, die für Werte des selben Vektors $\vec{a_k}$ stehen, einheitlich mit der Farbe C_k. Diese Konstruktion besitzt eine beschränkte Baumtiefe, denn alle Sterne an sich haben die konstante Baumtiefe 1, und da wir ein festes ℓ betrachten, ist die Baumtiefe der gesamten Struktur konstant und somit beschränkt.

Wir betrachten nun eine MSO-Formel $\varphi(X_1, \ldots, X_\ell)$, die so definiert ist, dass für $k \in \{1, \ldots, \ell\}$ die Relationsvariable X_k nur Sterne enthält, die mit der Farbe C_k gefärbt sind; dabei darf sie Sterne nur vollständig oder gar nicht enthalten. Wird in φ ein zu dem Wert $(\vec{a_k})_i$ korrespondierender Stern gewählt, so müssen auch alle Sterne, die zu Werten $(\vec{a_t})_i$ mit $t, k \in \{1, \ldots, \ell\}$ korrespondieren, gewählt werden. Wie φ für zwei Farben aussieht, ist in den Beweisen der Sätze 45 und 24 beschrieben, die Erweiterung auf ℓ-Farben verläuft äquivalent dazu.

Das Lösungshistogramm histogram(G, φ) beschreibt an Position $(i_1, \ldots, i_\ell)$, wie viele Möglichkeiten es gibt, um gleichzeitig i_1 Knoten der Farbe C_1, i_2 Knoten der Farbe C_2, …, i_ℓ Knoten der Farbe C_ℓ zu wählen. Um nun ℓ-UNARY-SUBSETSUM zu entscheiden, müssen wir prüfen, ob im Histogramm histogram$(G, \varphi)[s_1][s_2] \ldots [s_\ell] > 0$ gilt, und entsprechend akzeptieren. Nach Satz 12 können die entsprechenden Bits des Lösungshistogramms von mehreren TC^0-Schaltkreisen berechnet werden. Ein weiter TC^0-Schaltkreis prüft, ob mindestens einer dieser Einträge größer als 0 ist. Zusammen folgt somit ℓ-UNARY-SUBSETSUM $\in TC^0$.

Da UNARY-SUBSETSUM für $\ell = 1$ ein Spezialfall von ℓ-UNARY-SUBSETSUM ist, folgt die TC^0-Härte von ℓ-UNARY-SUBSETSUM. □

▶ Satz 30
Das Problem ℓ-PARTIAL-UNARY-SUBSETSUM ist NL-vollständig.

Beweis: Wir wollen zunächst ℓ-PARTIAL-UNARY-SUBSETSUM $\in$ NL zeigen. Eine nichtdeterministische Turingmaschine mit ℓ-Arbeitsbändern kann eine Eingabe für ℓ-PARTIAL-UNARY-SUBSETSUM lesen und dabei einige $i \in \{1, \ldots, p\}$ raten. Sie summiert dann für alle $k \in \{1, \ldots, \ell\}$ die $(\vec{a_k})_i$ auf Arbeitsband k auf und prüft für alle $i \in \{1, \ldots, p\}$, ob der Wert auf Arbeitsband k größer oder gleich dem Wert $(\vec{c_k})_i$ ist. Ist dies nicht der Fall, so verwirft die Turingmaschine sofort. Erreicht die Maschine das Ende der Eingabe, so prüft sie, ob der Wert auf Arbeitsband k gleich dem Wert s_k auf dem Eingabeband ist und akzeptiert, wenn dies für alle $k \in \{1, \ldots, \ell\}$ der Fall ist. Die Summationen sind in logarithmischem Platz möglich und da die Turingmaschine nichtdeterministisch ist, folgt ℓ-PARTIAL-UNARY-SUBSETSUM $\in$ NL.

Das Problem PARTIAL-UNARY-SUBSETSUM ist für $\ell = 1$ ein Spezialfall vom hier betrachteten ℓ-PARTIAL-UNARY-SUBSETSUM und dessen NL-Härte folgt somit sofort. □

Wir können ℓ-UNARY-SUBSETSUM weiter verallgemeinern, um die folgende Variante von LINEAR-PROGRAMMING zu erhalten:

▶ Problem 31 (ℓ-Linear-Programming)
Eingabe: Eingabevektoren $\vec{a_1}, \ldots, \vec{a_\ell}, \vec{c} \in \mathbb{N}^p$ mit $p \in \mathbb{N}$ sowie ein Zielvektor $\vec{b} \in \mathbb{N}^p$ und eine Zielsumme $s \in \mathbb{N}$.
Gesucht: Ein Vektor $\vec{x} \in \{0,1\}^p$, der das folgende Gleichungssystem löst:

$$\begin{aligned} \sum_{i=1}^{p} (\vec{a_k})_i \cdot x_i &\leqslant b_k \quad \text{für } k \in \{1, \ldots, \ell\}, \\ \sum_{i=1}^{p} c_i \cdot x_i &= s. \end{aligned}$$

Sprache: ℓ—LINEAR-PROGRAMMING $= \{\, \vec{a_1}, \ldots, \vec{a_\ell}, \vec{c}, \vec{b}$ in unärer Codierung | es existiert ein $\vec{x}$, sodass die obigen Gleichungen erfüllt sind $\}$. ◁

▶ Satz 32
Das Problem ℓ-LINEAR-PROGRAMMING ist TC^0-vollständig.

Beweis: Dass ℓ-LINEAR-PROGRAMMING $\in TC^0$ gilt, folgt aus den in diesem Abschnitt geführten Beweisen. Es handelt sich im Wesentlichen um eine Instanz von $(\ell+1)$-UNARY-SUBSETSUM mit der Verallgemeinerung, dass für ℓ Gleichungen keine Gleichheit, sondern nur eine Kleiner- oder Gleichbeziehung gelten muss. Diese Verallgemeinerung können wir lösen, indem wir das von uns beschriebenen Lösungsverfahren für ℓ-UNARY-SUBSETSUM leicht modifizieren (siehe Satz 29). Anstatt zu prüfen, ob im Lösungshistogramm der Eintrag $\text{histogram}(G, \varphi)[b_1] \ldots [b_\ell][s]$ größer 0 ist, prüfen wir nun, ob ein Index $(\nu_1, \ldots, \nu_\ell, s)$ mit $\nu_1 \leqslant b_1, \ldots, \nu_\ell \leqslant b_\ell$ existiert, sodass der Eintrag $\text{histogram}(G, \varphi)[\nu_1] \ldots [\nu_\ell][s]$ im Lösungshistogramm positiv ist.
Die TC^0-Härte und damit die TC^0-Vollständigkeit liefert uns eine Reduktion von UNARY-SUBSETSUM. Dazu übernehmen wir den Eingabevektor und die Zielsumme einer UNARY-SUBSETSUM-Instanz als $\vec{c}$ und s einer konstruierten Instanz von ℓ-LINEAR-PROGRAMMING. Setzen wir $\ell = 1$ und $\vec{a_1} = \vec{b} = \vec{0}$, folgt die Äquivalenz der Instanzen und somit die Aussage. □

4.2 Varianten von UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM mit geänderten Beschränkungen

Wir wissen, dass UNARY-SUBSETSUM ein TC^0-vollständiges Problem ist und dass Probleme wie PARTIAL-UNARY-SUBSETSUM und TUBE-UNARY-SUBSETSUM sogar NL-vollständig sind (siehe Abschnitt 3.1, 3.2 und 4.1). Der wesentliche Unterschied zwischen den von uns betrachteten TC^0-vollständigen Varianten von UNARY-SUBSETSUM und jenen, die NL-vollständig sind, ist eine obere

oder untere Schranke für die Wahl der Elemente. Im Folgenden schwächen wir diese Eigenschaft ab und fordern, dass nur noch an vorgegebenen Punkten, an den sogenanten *Wegpunkten*, ein Zielwert erreicht werden soll.

► Problem 33 (Waypoint-Subset-Sum)
Eingabe: Ein Eingabevektor $\vec{a} \in \mathbb{N}^p$ mit $p \in \mathbb{N}$ und ein Wegpunktvektor $\vec{w} \in (\mathbb{N} \cup \{\#\})^p$.
Gesucht: Ein Vektor $\vec{x} \in \{0,1\}$, sodass die folgende Gleichung gilt:

$$\sum_{i=1}^{\nu} x_i \cdot a_i = w_\nu \quad \text{für alle } \nu \in \{1,\dots,p\} \text{ mit } w_\nu \neq \#.$$

Sprache: WAYPOINT-UNARY-SUBSETSUM $= \{\, \vec{a}, \vec{w}$ in unärer Codierung $|$ es existiert ein $\vec{x}$, sodass die obigen Gleichungen erfüllt sind $\}$. ◁

► Satz 34
Das Problem WAYPOINT-UNARY-SUBSETSUM ist TC^0-vollständig.

Beweis: Wir zeigen zunächst unter Verwendung von Satz 12, dass das Problem WAYPOINT-UNARY-SUBSETSUM mit Hilfe eines TC^0-Schaltkreises entschieden werden kann. Dazu lösen wir die Teilabschnitte der Instanz separat, indem die Elemente bis zu einem Wegpunkt eine UNARY-SUBSETSUM-Instanz bilden und diese Elemente in der nächst höheren Instanz durch den Wegpunkt selbst ersetzt werden; in jeder Instanz erzwingen wir die Wahl des vorherigen Wegpunktes. Wir konstruieren für jedes Intervall $[i,j]$ mit $i = 1$ oder $w_i \neq \#$; $w_j \neq \#$ und $w_\nu = \#$ für alle $\nu \in \{i+1,\dots,j-1\}$ einen Graphen G_{ij}, welcher für jedes $k \in \{i+1,\dots,j\}$ einen Stern mit a_k Knoten, sowie einen $\square$-gefärbten Stern mit w_i Knoten enthält. Einen solchen Graphen G_{ij} beschreiben wir als logische Struktur $G_{ij} = (V, E^G, C_\square^G)$ über der logischen Signatur $\tau = (E^2, C_\square^1)$. Wir betrachten nun eine MSO-Formel $\varphi(X)$, die genau dann zu wahr ausgewertet wird, wenn die Relationsvariable X alle gefärbten Sterne vollständig und alle anderen Sterne entweder vollständig oder gar nicht enthält. Wir können $\varphi(X)$ formal beschreiben durch:

$$\varphi(X) = \forall x \,.\, \big(X(x) \to \forall y \,.\, E(x,y) \to X(y)\big) \wedge \big(C_\square(x) \to X(x)\big).$$

Da alle Sterne – und somit alle Graphen – eine konstante Baumtiefe besitzen, können wir nach Satz 12 die Bits der Lösungshistogramme histogram(G_{ij}, φ) jeweils mit TC^0-Schaltkreisen berechnen. Ein derartiges Lösungshistogramm histogram(G_{ij}, φ) ist ein eindimensionales Array, dessen i*ter* Eintrag angibt, wie viele Möglichkeiten es gibt, um in G_{ij} Sterne so zu wählen, dass alle gefärbten Knoten und insgesamt i Knoten gewählt werden. Um zu prüfen, ob ein Wegpunkt w_j erreicht werden kann, muss also ein TC^0-Schaltkreis prüfen, ob der letzte Eintrag von histogram(G_{ij}, φ) positiv ist. Da die Anzahl

der Wegpunkte polynomiell in der Eingabe beschränkt ist – und da die Wegpunkte im voraus bekannt sind – können TC^0-Schaltkreise alle Wegpunkte gleichzeitig prüfen und jeweils eine 1 ausgeben, wenn der entsprechende Wegpunkt erreicht werden kann. Ein weiterer TC^0-Schaltkreis kann nun die Ausgabe aller dieser TC^0-Schaltkreise prüfen und akzeptieren, wenn alle eine 1 ausgegeben. Es folgt WAYPOINT-UNARY-SUBSETSUM $\in TC^0$.

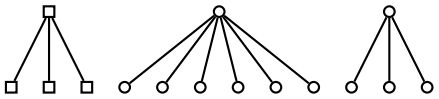
Besitzt der Wegpunktvektor die Form $\vec{w} = (\#, \ldots, \#, s)$ mit $s \in \mathbb{N}$, so entspricht die Instanz von WAYPOINT-UNARY-SUBSETSUMeiner Instanz des Problems UNARY-SUBSETSUM. Entsprechend ist UNARY-SUBSETSUM ein Spezialfall von WAYPOINT-UNARY-SUBSETSUM und es folgt die TC^0-Härte für das Problem WAYPOINT-UNARY-SUBSETSUM. □

► Beispiel 35

Wir betrachten eine Instanz von WAYPOINT-UNARY-SUBSETSUM, gegeben durch die Vektoren $\vec{a} = (1, 3, 7, 4)$ und $\vec{w} = (\#, 4, \#, 11)$. Der Wegpunktvektor enthält zwei Wegpunkte, wir müssen entsprechend einen Graphen für das Intervall $[1, 2]$ und einen für das Intervall $[2, 4]$ konstruieren. Wir betrachten zunächst den Graph zum Intervall $[1, 2]$ und das entsprechende Lösungshistogramm:

|X|	1	2	3	4
Anzahl der Lösungen	1	0	1	1

Für das Intervall $[2, 4]$ entsteht dann der folgende Graph mit einem gefärbten Stern, der den Wegpunkt der vorherigen Instanz repräsentiert und gewählt werden muss; im Lösungshistogramm sind – wegen der Übersichtlichkeit – nur positive Einträge abgebildet.



|X|	4	8	11	15
Anzahl der Lösungen	1	1	1	1

Die markierten Einträge in den Lösungshistogrammen zeigen, dass alle Wegpunkte erreicht werden können und folglich ist die hier betrachtete Instanz von WAYPOINT-UNARY-SUBSETSUM lösbar. ◁

Die Ergebnisse dieses Abschnittes geben Aufschluss über die Komplexität von PARTIAL-UNARY-SUBSETSUM, denn eine Instanz des hier betrachtete WAYPOINT-UNARY-SUBSETSUM sieht – für eine Wegpunktvektor, der keine # enthält – dem NL-vollständigen Problem PARTIAL-UNARY-SUBSETSUM sehr ähnlich und ist dennoch TC^0-vollständig: Für diesen Spezialfall ist der einzige Unterschied zwischen den Problemen die Größer- oder Gleichforderung der PARTIAL-UNARY-SUBSETSUM-Instanz, die die einfache Gleichheit in

WAYPOINT-UNARY-SUBSETSUM ersetzt. Diese ist auch der Grund, warum sich das beschriebene Lösungsverfahren für WAYPOINT-UNARY-SUBSETSUM nicht auf PARTIAL-UNARY-SUBSETSUM anwenden lässt; wir können bei einer Instanz von PARTIAL-UNARY-SUBSETSUM nicht einfach mit dem vorherigen Wegpunkt weiterrechnen, da wir nicht wissen, ob er mit Gleichheit erreicht oder drastisch überschritten wurde.

Wir haben mit PARTIAL-UNARY-SUBSETSUM eine Variante betrachtet, die das Ausgangsproblem UNARY-SUBSETSUM um eine untere Schranke erweitert. Mit dieser Erweiterung wurde das Problem deutlich komplexer und ließ sich nicht mehr mit einem TC^0-Schaltkreis lösen. Im Folgenden untersuchen wir diesen Zusammenhang genauer und beschränken UNARY-SUBSETSUM weiter, indem wir PARTIAL-UNARY-SUBSETSUM um eine obere Schranke erweitern.

► Problem 36 (Tube-Subset-Sum)
Eingabe: Drei Vektoren $\vec{a}, \vec{b}, \vec{c} \in \mathbb{N}^p$ mit $p \in \mathbb{N}$ sowie eine Zielsumme $s \in \mathbb{N}$.
Gesucht: Ein Vektor $\vec{x} \in \{0,1\}^p$, sodass Folgendes gilt:

$$\begin{aligned}
&\sum_{i=1}^{p} x_i \cdot a_i = s, \\
&\sum_{i=1}^{\nu} x_i \cdot a_i \geqslant b_\nu \quad \text{für alle } \nu \in \{1,\dots,p\}, \\
&\sum_{i=1}^{\nu} x_i \cdot a_i \leqslant c_\nu \quad \text{für alle } \nu \in \{1,\dots,p\}.
\end{aligned}$$

Sprache: $\text{TUBE-UNARY-SUBSETSUM} = \{\, 0^{a_1}10^{b_1}10^{c_1}11\dots110^{a_p}10^{b_p}10^{c_p}1110^s \mid \exists I \subseteq \{1,\dots,p\} \centerdot \sum_{i\in I} a_i = s \wedge \forall \nu \leqslant p \centerdot \sum_{i\in I, i\leqslant \nu} a_i \geqslant b_\nu \wedge \sum_{i\in I, i\leqslant \nu} a_i \leqslant c_\nu \,\}.$ ◁

► Satz 37
Das Problem TUBE-UNARY-SUBSETSUM ist NL-vollständig.

Beweis: Wir betrachten eine Instanz von TUBE-UNARY-SUBSETSUM der Form

$$0^{a_1}10^{b_1}10^{c_1}11\dots110^{a_p}10^{b_p}10^{c_p}1110^s.$$

Eine nichtdeterministische Turingmaschine kann diese Eingabe von links nach rechts lesen und dabei einige a_i mit $i \in \{1,\dots,p\}$ raten und auf ihrem Arbeitsband aufsummieren. Wann immer die Turingmaschine ein b_i erreicht, prüft sie, ob der Wert auf ihrem Arbeitsband größer als b_i ist. Immer, wenn sie ein c_i erreicht, prüft sie, ob der Wert auf ihrem Arbeitsband kleiner als c_i ist. Ist dies einmal nicht der Fall, verwirft sie sofort. Erreicht sie das Ende der Eingabe, so prüft sie, ob der Wert auf ihrem Arbeitsband dem Wert s auf dem Eingabeband entspricht, und akzeptiert, wenn dies der

Fall ist. Da die Eingabe unär codiert ist, reicht logarithmischer Platz für die Aufsummierung. Da die Turingmaschine nichtdeterministisch ist, folgt die Korrektheit des Algorithmus und folglich TUBE-UNARY-SUBSETSUM $\in$ NL.

Um die NL-Härte von TUBE-UNARY-SUBSETSUM zu zeigen, führen wir eine Reduktion von PARTIAL-UNARY-SUBSETSUM. Dazu übernehmen wir die Eingabe einer Instanz von PARTIAL-UNARY-SUBSETSUM als Eingabe für eine TUBE-UNARY-SUBSETSUM-Instanz und setzen zusätzlich alle Elemente der oberen Schranke auf die Zielsumme. Entsprechend sind die unteren Schranken beider Instanzen identisch und die untere Schranke der Instanz von TUBE-UNARY-SUBSETSUM wird genau dann unterschritten, wenn die untere Schranke der PARTIAL-UNARY-SUBSETSUM-Instanz unterschritten wird. Da die Eingabe nur aus natürlichen Zahlen besteht, kann der Wert der Berechnung nie größer als die Zielsumme werden, wenn diese insgesamt erreicht wird, und folglich wird die obere Schranke von TUBE-UNARY-SUBSETSUM nicht überschritten, wenn die Instanz von PARTIAL-UNARY-SUBSETSUM lösbar ist. Zusammen folgt, dass die TUBE-UNARY-SUBSETSUM-Instanz genau dann lösbar ist, wenn die Instanz von PARTIAL-UNARY-SUBSETSUM lösbar ist. □

Bei dem aus Abschnitt 4.1 bekannten Problem UNARY-KNAPSACK (Problem 22) besitzen die Elemente der Eingabe einen Wert und ein Gewicht, gesucht ist eine Lösung, die einen möglichst hohen Wert besitzt und dabei das maximale Transportgewicht nicht überschreitet. Dieser Ansatz ist sinnvoll, wenn wir jedem Element einen Wert und ein Gewicht zuordnen können; ist das Gewicht jedoch genormt oder von geringem Interesse, so bringt uns dieser Ansatz wenig. Betrachten wir zum Beispiel folgendes illustriertes Szenario eines Containerschiffes, das im Hafen liegt und beladen werden soll. Die Reederei möchte natürlich mit möglichst wenig Fahrten einen möglichst hohen Gewinn erwirtschaften, weshalb Container mit einem möglichst hohen Wert verladen werden sollen. Die grundlegende Idee von Containern ist, dass ihre Größe genormt ist und dass ihr Gewicht beim Verladen auf ein Schiff nur eine untergeordnete Rolle spielt. Entscheidend ist, dass auf einem Containerschiff nur Platz für k Container ist, unabhängig davon, wie schwer diese sind. Unsere Aufgabe ist es nun, aus allen Containern, die der Reederei im Hafen zur Verfügung stehen, k auszuwählen und zu verladen, so dass die Summe der Werte dieser Container möglichst groß ist. Formal lässt sich dieses Problem wie folgt beschreiben:

▶ Problem 38 (Choose-k-Subset-Sum)
Eingabe: Ein Vektor $\vec{a} \in \mathbb{N}^p$ mit $p \in \mathbb{N}$ sowie die Zahlen $s, k \in \mathbb{N}$.
Gesucht: Ein Vektor $\vec{x} \in \{0,1\}^p$, sodass gilt:

$$\sum_{i=1}^{p} x_i \cdot a_i = s \quad \text{und} \quad \sum_{i=1}^{p} x_i \leqslant k.$$

Sprache: CHOOSE-K-UNARY-SUBSETSUM $= \{ 0^{a_1}1 \ldots 10^{a_p}110^s10^k \mid$
$\exists I \subseteq \{1, \ldots, p\} \,.\, \sum_{i \in I} a_i = s \wedge |I| \leqslant k \}.$ ◁

Bei der Seefahrt herrschen strenge Sicherheitsregeln und jedes Schiff darf nur so stark beladen werden, dass es einen gewissen Tiefgang nicht überschreitet. Nun sind die Container genormt und eigentlich sollten alle k Container problemlos verladen werden können, dennoch kann es passieren, dass ein paar Container einmal überdurchschnittlich schwer sind. In diesem Fall erreicht das Schiff schon während des Verladens der Container den kritischen Tiefgang, weshalb nicht alle Container verladen werden können. Eine professionelle Reederei möchte daher zunächst jene Container verladen, die einen möglichst hohen Wert haben, sodass der Verlust nicht zu groß ist, wenn die letzten Container nicht mehr verladen werden dürfen. Das so entstehende Problem lässt sich formal wie folgt beschreiben:

▶ Problem 39 (Choose-k-Partial-Subset-Sum)
Eingabe: Die Vektoren $\vec{a}, \vec{c} \in \mathbb{N}^p$ mit $p \in \mathbb{N}$ sowie die Zahlen $s, k \in \mathbb{N}$.
Gesucht: Ein Vektor $\vec{x} \in \{0,1\}^p$, sodass gilt:

$$\sum_{i=1}^{p} x_i \cdot a_i = s \quad \text{und} \quad \sum_{i=1}^{p} x_i \leqslant k,$$
$$\sum_{i=1}^{\nu} x_i \cdot a_i \geqslant c_\nu \quad \text{für alle } \nu \in \{1, \ldots, p\}.$$

Sprache: CHOOSE-K-PARTIAL-UNARY-SUBSETSUM $= \{$
$0^{a_1}10^{c_1}11 \ldots 110^{a_p}10^{c_p}1110^s10^k \mid$
$\exists I \subseteq \{1, \ldots, p\} \,.\, \sum_{i \in I} a_i = s \wedge |I| \leqslant k$
$\wedge \forall_{\nu \leqslant p} \,.\, \sum_{i \in I, i \leqslant \nu} a_i \geqslant c_\nu \}.$ ◁

▶ Satz 40

Das Problem CHOOSE-k-UNARY-SUBSETSUM ist TC^0-vollständig.

Beweis: Zunächst zeigen wir CHOOSE−k−UNARY-SUBSETSUM $\in TC^0$ und betrachten dazu die Eingabe einer Instanz von CHOOSE-k-UNARY-SUBSETSUM als Graph G, indem wir für alle $i \in \{1, \ldots, p\}$ einen Stern mit a_i Knoten, dessen Zentralknoten wir mit $\square$ färben, konstruieren. Diesen Graphen können wir als logische Struktur $G = (V, E^G, C_\square^G)$ über der Signatur $\tau_G = (E^2, C_\square^1)$ darstellen. Wir betrachten die MSO-Formel $\varphi(X, Y)$, die wie folgt definiert ist:

$$\varphi(X, Y) = \forall x \,.\, \big(X(x) \to \forall y \,.\, E(x, y) \to X(y)\big) \wedge \big(Y(x) \leftrightarrow C_\square(x) \wedge X(x)\big).$$

Diese Formel wird zu wahr ausgewertet, wenn X Sterne vollständig enthält und Y einen Knoten für jeden Stern in X enthält. Somit beschreibt die Kardinalität von X eine mögliche Zielsumme, wenn die Anzahl der gewählten Elemente der Kardinalität von Y entspricht. Das Lösungshistogramm $\text{histogram}(G, \varphi)$ beschreibt entsprechend an Position (x, y) die Anzahl der Möglichkeiten, um die Summe x unter Verwendung von y Elementen zu erreichen. Um CHOOSE-k-UNARY-SUBSETSUM zu lösen, müssen wir demnach prüfen, ob ein $\nu \leqslant k$ existiert, sodass $\text{histogram}(G, \varphi)[s][\nu] > 0$ gilt. Da wir das Histogramm nach Satz 12 mit TC^0-Schaltkreisen bitweise berechnen können, folgt CHOOSE−k−UNARY-SUBSETSUM $\in TC^0$.

Wir reduzieren von einer UNARY-SUBSETSUM-Instanz, um die TC^0-Härte von CHOOSE-k-UNARY-SUBSETSUM zu zeigen. Dazu übernehmen wir die Eingabe einer UNARY-SUBSETSUM-Instanz direkt als Eingabe für eine Instanz von CHOOSE-k-UNARY-SUBSETSUM und setzen $k = p$, wenn p die Länge des Eingabevektors der UNARY-SUBSETSUM-Instanz ist. Durch $k = p$ sind die Instanzen von UNARY-SUBSETSUM und CHOOSE-k-UNARY-SUBSETSUM äquivalent und es folgt die Aussage. □

► Beispiel 41

Wir betrachten im Folgenden eine Instanz von dem Problem CHOOSE-k-UNARY-SUBSETSUM mit einem Wertevektor $\vec{a} = \{3, 6, 5, 1\}$, der Zielsumme $s = 10$ und der Anzahl möglicher Elemente $k = 3$. Der zu dieser Instanz korrespondierende Graph ist in der folgenden Abbildung dargestellt. In diesem Graphen sind die Zentralknoten der Sterne mit □ gekennzeichnet.

$\lvert X\rvert, \lvert Y\rvert$	1	2	3	4
1	1	0	0	0
2	0	0	0	0
3	1	0	0	0
4	0	1	0	0
5	1	0	0	0
6	1	1	0	0
7	0	1	0	0
8	0	1	0	0
9	0	1	1	0
10	0	0	1	0
11	0	1	0	0
12	0	0	1	0
13	0	0	0	0
14	0	0	1	0
15	0	0	0	1

In der Tabelle auf der rechten Seite ist das zu dieser CHOOSE-k-UNARY-SUBSETSUM-Instanz korrespondierende Lösungshistogramm dargestellt. In diesem gibt der Eintrag (x, y) an, wie viele Möglichkeiten es gibt, um y Elemente so zu wählen, dass ihre Summe x ergibt. Aus der markierten Zeile können wir entnehmen, dass die Instanz von CHOOSE-k-UNARY-SUBSETSUM lösbar ist, denn die Zielsumme $s = 10$ ist mit 3 Elementen erreichbar. Würden wir dieselbe Instanz für $k = 2$ betrachten, so wäre sie nicht lösbar. ◁

► Satz 42

Das Problem CHOOSE-k-PARTIAL-UNARY-SUBSETSUM ist NL-vollständig.

Beweis: Wir betrachten eine Instanz von CHOOSE-k-PARTIAL-UNARY-SUBSETSUM der Form

$$0^{a_1}10^{c_1}11\ldots110^{a_p}10^{c_p}1110^{s}10^{k}.$$

Eine nichtdeterministische Turingmaschine mit zwei Arbeitsbändern liest diese Eingabe von links nach rechts und rät dabei einige $i \in \{1, \ldots, p\}$, für die sie auf ihrem ersten Arbeitsband das a_i aufsummiert. Für jedes gewählte a_i inkrementiert sie einen Zähler auf dem zweiten Arbeitsband. Wann immer die Turingmaschine ein c_i für $i \in \{1, \ldots, p\}$ erreicht, prüft sie, ob der Wert auf ihrem ersten Arbeitsband größer gleich dem c_i auf dem Eingabeband ist und verwirft, wenn dies nicht der Fall ist. Erreicht sie das Ende der Eingabe, so prüft sie, ob der Wert auf dem ersten Arbeitsband gleich s und ob der Wert auf dem zweiten Arbeitsband kleiner oder gleich k ist. Wenn beides gilt, akzeptiert die Turingmaschine. Der Zähler ist durch p beschränkt und die Summation der a_i ist in logarithmischem Platz möglich, da die Eingabe unär codiert ist.

Um die TC^0-Härte von CHOOSE-k-PARTIAL-UNARY-SUBSETSUM zu zeigen, reduzieren wir von PARTIAL-UNARY-SUBSETSUM. Dazu übernehmen wir die

Eingabe einer PARTIAL-UNARY-SUBSETSUM-Instanz direkt als Eingabe für eine konstruierte Instanz von CHOOSE-k-PARTIAL-UNARY-SUBSETSUM und setzen $k = p$, wobei p die Länge des Eingabevektors der PARTIAL-UNARY-SUBSETSUM-Instanz ist. Wegen $k = p$ sind dann beide Instanzen äquivalent und es folgt die Aussage. □

4.3 Varianten von UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM über anderen Körpern

Bisher haben wir immer Eingaben über dem Körper der natürlichen Zahlen betrachtet. In diesem Abschnitt wollen wir nun die Auswirkungen auf die Komplexität von UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM untersuchen, wenn wir den Körper, über dem die Eingabe definiert ist, wechseln. Zunächst wollen wir die Eingabe der Probleme UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM von den natürlichen Zahlen auf die ganzen Zahlen verallgemeinern und die entstehenden Probleme untersuchen.

► Problem 43 (Integer-Subset-Sum)
Eingabe: Ein Vektor $\vec{a} \in \mathbb{Z}^p$ mit $p \in \mathbb{N}$ sowie eine Zielsumme $b \in \mathbb{Z}$.
Gesucht: Ein Vektor $\vec{x} \in \{0,1\}^p$, sodass folgende Gleichung gilt:

$$\sum_{i=1}^{p} a_i \cdot x_i = b.$$

Sprache: Es sei σ das Vorzeichen der folgenden Zahl, wobei $\sigma = 1$ gilt, falls diese positiv ist, und sonst $\sigma = 2$.
INTEGER-UNARY-SUBSETSUM $= \{\, 0^{\sigma_1}10^{a_1}11\ldots110^{\sigma_p}10^{a_p}1110^{\sigma_b}10^{b} \mid \exists I \subseteq \{1,\ldots,p\}.\ \sum_{i\in I} a_i = b \,\}$. ◁

► Problem 44 (Partial-Integer-Subset-Sum)
Eingabe: Zwei Vektoren $\vec{a}, \vec{c} \in \mathbb{Z}^p$ mit $p \in \mathbb{N}$ sowie eine Zielsumme $b \in \mathbb{Z}$.
Gesucht: Ein Vektor $\vec{x} \in \{0,1\}^p$, sodass die folgenden Gleichungen erfüllt sind:

$$\sum_{i=1}^{p} x_i \cdot a_i = b,$$
$$\sum_{i=1}^{\nu} x_i \cdot a_i \geqslant c_\nu \quad \text{für alle } \nu \in \{0,\ldots,p\}.$$

Sprache: Es sei σ das Vorzeichen der folgenden Zahl, wobei $\sigma = 1$ gilt, falls diese positiv ist, und sonst $\sigma = 2$.
PARTIAL-INTEGER-UNARY-SUBSETSUM $= \{$
$0^{\sigma}10^{a_1}110^{\sigma}10^{c_1}111\ldots1110^{\sigma}10^{a_p}110^{\sigma}10^{c_p}11110^{\sigma}10^{b} \mid$
$\exists I \subseteq \{1,\ldots,p\}.\ \sum_{i\in I} a_i = b \wedge \forall_{\nu\leqslant p} \sum_{i\in I, i\leqslant\nu} a_i \geqslant c_\nu \,\}$. ◁

Wir wollen zunächst die Komplexität von INTEGER-UNARY-SUBSETSUM untersuchen.

► Satz 45
Das Problem INTEGER-UNARY-SUBSETSUM ist TC^0-vollständig.

Beweis: Wir wollen eine Instanz von INTEGER-UNARY-SUBSETSUM als Graphenstruktur mit beschränkter Baumtiefe darstellen. Um im Graphen sowohl positive als auch negative Zahlen durch Sterne darstellen zu können, betrachten wir einen Graphen mit Knotenfärbung, sodass die Menge der Knoten in zwei disjunkte Knotenmengen zerfällt. Wir können diesen Graphen über der Signatur $\tau = (E^2, C^1_{\oplus}, C^1_{\ominus})$ mit der Struktur $G = (V, E^G, C^G_{\oplus}, C^G_{\ominus})$ darstellen, wobei V die Knotenmenge, E^G die Kantenrelation zwischen zwei Knoten und $C^G_{\oplus}$, $C^G_{\ominus}$ die Färbung eines Knotens mit $\oplus$ bzw. $\ominus$ beschreibt. Wir erzeugen nun für jedes $i \in \{1, \ldots, p\}$ einen Stern mit a_i Knoten. Ist a_i positiv, so färben wir den Stern mit $\oplus$, andernfalls färben wir ihn mit $\ominus$. Wir betrachten nun eine logische Formel $\varphi(X, Y)$, die genau dann zu wahr ausgewertet werden soll, wenn die Mengenvariable X nur $\oplus$-Sterne vollständig oder gar nicht enthält und die Mengenvariable Y entsprechend nur $\ominus$-Sterne vollständig oder gar nicht enthält. Dazu definieren wir $\varphi(X, Y)$ wiefolgt:

$$\varphi(X,Y) = \forall x \,.\, \big(X(x) \to C_{\oplus}(x)\big) \wedge \big(Y(x) \to C_{\ominus}(x)\big) \\ \wedge \,.\, X(x) \vee Y(x) \to \forall y \,.\, E(x,y) \to X(y) \vee Y(y).$$

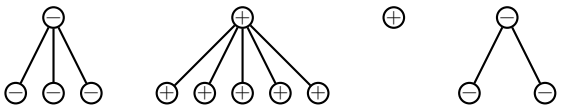
Wir können jede ganze Zahl $b \in \mathbb{Z}$ als Differenz zweier natürlicher Zahlen $x, y \in \mathbb{N}$ darstellen, indem wir $b = x - y$ setzen. Das Lösungshistogramm $\text{histogram}(G, \varphi)$ ist ein zweidimensionales Array, dessen Eintrag an der Position (x, y) größer als 0 ist, wenn wir x verschiedene $\oplus$-Knoten und y verschiedene $\ominus$-Knoten so wählen können, dass die entsprechenden Sterne vollständig gewählt sind. Da x für positive und y für negative Werte steht, entspricht dieser Eintrag dem Wert $x - y$. Wir müssen nun also prüfen, ob ein positiver Eintrag $\text{histogram}(G, \varphi)[i][j]$ mit $b = i - j$ existiert. Für eine Instanz mit n positiven und m negativen Knoten müssen wir dazu im Lösungshistogramm der Größe $n \times m$ die Werte der Diagonalen (a_{ij}) mit $i = b, b+1, \ldots, m$ und $j = 0, 1, \ldots, m - b$, wenn b positiv ist, und sonst die Werte der Diagonalen (a_{kl}) mit $k = 0, 1, \ldots, n - b$ und $l = b, b+1, \ldots, n$ berechnen. Dies sind höchstens $\min(n, m)$ viele Werte, deren Bits von TC^0-Schaltkreisen berechnet werden können. Ein weiterer TC^0-Schaltkreis kann nun mit einem Threshold-Gatter prüfen, ob eines dieser Bits auf 1 gesetzt ist. Diese Berechnung ist nach Satz 11 möglich, denn alle Sterne und damit auch der Graph an sich besitzen die konstante Baumtiefe 2. Es gilt folglich INTEGER-UNARY-SUBSETSUM $\in TC^0$.

Um die TC^0-Härte zu zeigen, reduzieren wir von UNARY-SUBSETSUM. Von einer gegebenen UNARY-SUBSETSUM-Instanz übernehmen wir sowohl den Ein-

gabevektor als auch die Zielsumme als Eingabe für eine Instanz vom Problem INTEGER-UNARY-SUBSETSUM. Beide Instanzen sind dann identisch, denn in der Instanz von INTEGER-UNARY-SUBSETSUM kommen keine negativen Zahlen vor. Entsprechend ist die konstruierte Instanz genau dann lösbar, wenn auch die UNARY-SUBSETSUM-Instanz lösbar ist. Die oben beschriebene Reduktion ist mit einer FO-Reduktion, wie wir sie in Abschnitt 3.1 geführt haben, möglich. □

► Beispiel 46

Wir betrachten eine Instanz von INTEGER-UNARY-SUBSETSUM mit der Eingabe $\vec{a} = \{-4, 6, 1, -3\}$ und $b = 2$. Dazu konstruieren wir einen entsprechenden Graphen mit beschränkter Baumtiefe, wie in der folgenden Abbildung zu sehen. Sterne für positive Werte sind mit ⊕-Knoten und negative Sterne durch ⊖-Knoten gekennzeichnet.



Das Lösungshistogramm $\text{histogram}(G, \varphi)$ zu dieser Instanz ist in der folgenden Tabelle abgebildet. Die Diagonale mit möglichen Zerlegungen von $b = 2$ ist markiert. Da in der Diagonalen ein positiver Eintrag an der Position $(6, 4)$ existiert, können wir dem Lösungshistogramm entnehmen, dass diese Instanz von INTEGER-UNARY-SUBSETSUM eine Lösung besitzt.

|X|, |Y|	0	1	2	3	4	5	6	7
0	0	0	0	1	1	0	0	1
1	1	0	0	1	1	0	0	1
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	1	0	0	1	1	0	0	1
7	1	0	0	1	1	0	0	1

◁

Somit haben wir gezeigt, dass sich die Komplexität von UNARY-SUBSETSUM nicht ändert, wenn wir in der Eingabe nicht nur natürliche, sondern auch ganze Zahlen erlauben. Im folgenden untersuchen wir nun, ob sich eventuell die Komplexität von PARTIAL-INTEGER-UNARY-SUBSETSUM ändert, wenn wir hier ganze Zahlen in der Eingabe zulassen.

► Satz 47
Das Problem PARTIAL-INTEGER-UNARY-SUBSETSUM ist NL-vollständig.

Beweis: Wir betrachten eine Instanz von PARTIAL-INTEGER-UNARY-SUBSETSUM der Form

$$0^{\sigma}10^{a_1}110^{\sigma}10^{c_1}111\ldots1110^{\sigma}10^{a_p}110^{\sigma}10^{c_p}11110^{\sigma}10^{b}.$$

Dabei beschreibt $\sigma \in \{1,2\}$ das Vorzeichen und es gilt $\sigma = 1$, wenn der folgende Wert positiv ist und $\sigma = 2$, wenn er negativ ist. Eine nichtdeterministische Turingmaschine kann diese Eingabe von links nach rechts lesen und dabei einige $i \in \{1,\ldots,p\}$ raten. Für jedes geratene i addiert oder subtrahiert sie – je nach Vorzeichen – a_i auf den Wert auf ihrem Arbeitsband. Wann immer die Turingmaschine ein c_ν für $\nu \in \{1,\ldots,p\}$ erreicht, prüft sie, ob der Wert auf ihrem Arbeitsband größer gleich diesem c_ν ist und berücksichtigt dabei, ob c_ν positiv oder negativ ist. Ist der Wert einmal nicht größer, so werwirft die Turingmaschine sofort. Erreicht sie das Ende der Eingabe, prüft sie, ob b positiv oder negativ ist und vergleicht dann den gespeicherten Wert mit b. Da dessen Codierungslänge logarithmisch in der Eingabe beschränkt ist, folgt, dass die Turingmaschine mit logarithmischem Platz auskommt und folglich PARTIAL-INTEGER-UNARY-SUBSETSUM $\in$ NL.

Um die Härte zu zeigen, wollen wir von PARTIAL-UNARY-SUBSETSUM reduzieren und übernehmen dazu die Eingabe von PARTIAL-UNARY-SUBSETSUM direkt als Eingabe für eine Instanz von PARTIAL-INTEGER-UNARY-SUBSETSUM. Die Instanzen und Lösungen sind dann, abgesehen von der entsprechenden Codierung, identisch, da keine negativen Zahlen vorkommen und folglich ist PARTIAL-INTEGER-UNARY-SUBSETSUM genau dann lösbar, wenn die Instanz von PARTIAL-UNARY-SUBSETSUM lösbar ist; es folgt die Aussage. □

Wir haben gezeigt, dass sich die Komplexitäten von UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM nicht verändern, wenn wir Eingaben über den ganzen Zahlen betrachten. Wir erweitern die Eingabe nun weiter auf die rationalen Zahlen und untersuchen die Komplexität der entstehenden Variante von UNARY-SUBSETSUM.

► Problem 48 (ℚ-Subset-Sum)
Eingabe: Ein Vektor $\vec{a} \in \mathbb{Q}^p$ mit $p \in \mathbb{N}$ sowie eine Zielsumme $b \in \mathbb{Q}$.
Gesucht: Ein Vektor $x \in \{0,1\}^p$, sodass folgende Gleichung gilt:

$$\sum_{i=1}^{p} x_i \cdot a_i = b.$$

Sprache: Q-UNARY-SUBSETSUM $= \{\, \vec{a}, b$ in unärer Codierung $\mid$
$\exists I \subseteq \{1,\ldots,p\} \,.\, \sum_{i\in I} a_i = b \,\}$. ◁

Unsere Untersuchungen von Q-UNARY-SUBSETSUM zeigen, dass sich dieses Problem unerwartet schwer entscheiden und in eine Komplexitätsklasse ordnen lässt. Zwar sieht Q-UNARY-SUBSETSUM dem Problem UNARY-SUBSETSUM sehr ähnlich, doch ist es im Rahmen dieser Arbeit trotz vieler Bemühungen nicht gelungen, eine Aussage über die Vollständigkeit des Problems für eine Komplexitätsklasse zu formulieren. Wir können dennoch eine Härteaussage zeigen und anschließend die entstandenen Probleme diskutieren.

▶ Lemma 49
Das Problem Q-UNARY-SUBSETSUM ist NL-hart.

Beweis: Um dies zu zeigen, reduzieren wir von einer DAG-REACH-Instanz auf Q-UNARY-SUBSETSUM. Sei dazu $G = (V, E^G)$ mit Knoten $V = \{1, \ldots, q\}$ und Kanten $(i, j) \in E(G)$ eine DAG-REACH-Instanz. Da G zyklenfrei ist, können wir mit einer topologischen Sortierung die Knoten so beschriften, dass für alle Kanten $(i, j) \in E(G)$ stets $i < j$ gilt. Enthält G eine Kante a_{1q}, so reduzieren wir die Instanz von DAG-REACH auf eine triviale immer lösbare Instanz von Q-UNARY-SUBSETSUM – wie zum Beispiel eine die nur 0*en* enthält.

Für andere DAG-REACH-Instanzen weisen wir den Knoten $v \in \{2, \ldots, q-1\}$ Primzahlen p_{v-1} zu, wobei p_i die i*te* Primzahl ist. Dies ist möglich, denn für unäre Zahlen kann eine nichtdeterministische Logspace-Turingmaschine einen Primzahltest durchführen, indem sie die Eingabe binärcodiert auf ihrem Arbeitsband speichert und Bitweise prüft, ob diese Zahl durch eine entsprechende Zahl teilbar ist. Da des Weiteren für $i \geqslant 6$ die i*te* Primzahl nach dem Satz von Rosser und Schoenfeld durch $p_i < i(\log(i) + \log\log(i))$ beschränkt ist, können wir eine nichtdeterministische Logspace-Turingmaschine konstruieren, die die i*te* Primzahl berechnet [13].

Wir konstruieren nun für jede Kante $(i, j) \in E(G)$ des Graphen einen Wert a_{ij}. Dabei setzen wir für $i, j \in \{2, \ldots, q-1\}$ den Wert auf:

$$a_{ij} = -\frac{1}{p_{i-1}} + \frac{1}{p_{j-1}} < 0. \tag{i}$$

Weiter setzen wir für $i = 1$ den Wert auf:

$$a_{1j} = 1 + \frac{1}{p_{j-1}} > 1 \tag{ii}$$

und für $j = q$ auf:

$$a_{iq} = -\frac{1}{p_{i-1}} < 0. \tag{iii}$$

Wir sortieren die a_{ij} lexikografisch und verwenden sie als Eingabevektor für eine Q-UNARY-SUBSETSUM-Instanz; weiter setzen wir die Zielsumme $b = 1$ und zeigen, dass diese Instanz von Q-UNARY-SUBSETSUM genau dann lösbar ist, wenn die Instanz von DAG-REACH lösbar ist.

Nehmen wir zunächst an, dass die Instanz von DAG-REACH lösbar ist. Dann existiert in G ein Pfad

$$1 = i_0 \to i_1 \to \cdots \to i_t = q.$$

Somit gilt:

$$\begin{aligned}\sum_{\nu=0}^{t-1} a_{i_\nu i_{\nu+1}} &= a_{i_0 i_1} + a_{i_1 i_2} + \cdots + a_{i_{t-1} i_t} \\ &= 1 + \underbrace{\frac{1}{p_{i_1}-1} - \frac{1}{p_{i_1}-1} + \frac{1}{p_{i_2}-1} - \cdots - \frac{1}{p_{i_{t-1}}-1}}_{=0} \\ &= 1 = b\end{aligned}$$

und entsprechend ist die Instanz von Q-UNARY-SUBSETSUM lösbar.

Nehmen wir nun an, dass die Instanz von DAG-REACH nicht lösbar ist – ein wie oben beschriebener Pfad existiert also nicht. Für einen Widerspruch nehmen wir weiter an, dass die Instanz von Q-UNARY-SUBSETSUM trotzdem lösbar ist und die folgende – lexikografisch sortierte – Auswahl

$$I = \{\, a_{i_0 j_0}, a_{i_1 j_1}, a_{i_2 j_2}, \ldots, a_{i_m j_m} \,\}$$

existiert, sodass

$$\sum_{\nu=0}^{m} a_{i_\nu j_\nu} = 1 = b$$

gilt. In dieser Auswahl muss $i_0 = 1$ gelten, denn wegen (i) und (iii) würde anderenfalls

$$\sum_{\nu=0}^{m} a_{i_\nu j_\nu} < 0 \neq b$$

gelten. Per Induktion über die Anzahl n der ersten n Elemente von I können wir nun die Elemente der Auswahl bestimmen.

IA: Für $n = 1$ gilt:

$$\sum_{\nu=0}^{0} a_{i_\nu j_\nu} = a_{i_0 j_0} = a_{1 j_0} = 1 + \frac{1}{p_{j_0-1}}.$$

IV: Für beliebige, aber feste $n < m$ gilt:

$$\sum_{\nu=0}^{n} a_{i_\nu j_\nu} = 1 + \frac{1}{p_{j_n-1}}.$$

IS: Nach der Summation von n Elementen aus I gibt die Summe nach IV

$$\sum_{\nu=0}^{n} a_{i_\nu j_\nu} = 1 + \frac{1}{p_{j_n - 1}}$$

und insgesamt muss gelten

$$\sum_{\nu=0}^{m+1} a_{i_\nu j_\nu} = 1 = b.$$

Da p_{j_n-1} eine Primzahl ist, muss jede Menge von Brüchen, die in der Summe $-\frac{1}{p_{j_n-1}}$ ergeben soll, auch $\frac{1}{p_{j_n-1}}$ enthalten. Da die Auswahl sortiert ist, muss des Weiteren das $(n+1)$*te* Element somit $a_{j_n j_{n+1}}$ sein, also $i_{n+1} = j_n$. Wir unterscheiden nun zwei Fälle, sei dafür zunächst $j_{n+1} \neq q$ (also $n < m$), dann folgt

$$\sum_{\nu=0}^{n+1} a_{i_\nu j_\nu} = 1 + \frac{1}{p_{j_{n+1}-1}}.$$

Sei nun $j_{n+1} = q$ (also $n = m$), dann folgt

$$\sum_{\nu=0}^{n+1} a_{i_\nu j_\nu} = 1.$$

In diesem Fall bildet die Folge

$$a_{1 j_0} = a_{i_0 j_0} \to a_{i_1 j_1} \to \cdots \to a_{i_m j_m} = a_{i_m q}$$

einen Pfad von 1 nach q im Graphen.

Aus der Induktion folgt, dass die Auswahl im Graphen einen Pfad von 1 nach q aufspannt, wenn sie in der Summe 1 bildet. Dies ist ein Widerspruch zur Voraussetzung und folglich ist die Instanz von Q-UNARY-SUBSETSUM nicht lösbar, wenn die Instanz von DAG-REACH nicht löstbar ist. □

Mit der NL-Härte wissen wir, dass wir Q-UNARY-SUBSETSUM unter Standardannahmen nicht in eine Komplexitätsklasse kleiner als NL einordnen können. Es ist in dieser Arbeit auch nicht gelungen, einen NL-Algorithmus anzugeben, der das Problem Q-UNARY-SUBSETSUM löst. Das Problem eines klassischen Algorithmus – der über die Eingabe iteriert, Elemente rät und diese aufsummiert – ist die Größe der Zahlen, die bei der Berechnung auftreten können, denn für jede geratene Zahl muss – bei einer typischen Vorgehensweise – der gespeicherte Bruch mit dem geratene Bruch gleichnamig gemacht werden. Sind die geratenen Brüche alle teilerfremd, so wächst die Zahlendarstellung von Nenner und Zähler zu schnell und lässt sich nicht in

logarithmischem Platz speichern. Dies liegt daran, dass die Darstellung des Ergebnisses der Multiplikation von polynomiell vielen unären Zahlen exponentiell wächst und somit, selbst wenn das Ergebnis binär gespeichert wird, nicht in logarithmischem Platz unterzubringen ist. Allerdings lässt sich diese Zahl in polynomiellem Platz speichern, weshalb wir Hoffnung in P- oder NP-Algorithmen setzen können. Das Wachstum der Darstellung der Brüche ist allerdings auch dafür verantwortlich, dass es im Rahmen dieser Arbeit nicht gelungen ist, einen P-Algorithmus zu finden. Als obere Schranke können wir aber angeben, dass sich Q-UNARY-SUBSETSUM mit einem NP-Algorithmus lösen lässt.

► Satz 50
Es gilt Q-UNARY-SUBSETSUM $\in$ NP.

Beweis: Folgender NP-Algorithmus löst Q-UNARY-SUBSETSUM. Sei $\vec{a} \in \mathbb{Q}^p$ und $s \in \mathbb{Q}$ die unärcodierte Eingabe einer Q-UNARY-SUBSETSUM-Instanz. Eine nichtdeterministische Turingmaschine liest den Eingabevektor und rät einige $i \in \{1, \ldots, p\}$; die entsprechenden a_i speichert sie binärcodiert auf ihrem Arbeitsband. Dabei addiert sie das jeweils erratene a_i auf den aktuellen Wert auf ihrem Arbeitsband; das Ergebnis der dazu nötigen Multiplikation ist in der binären Codierung polynomiell in der Eingabe und lässt sich daher auf dem Band speichern. Hat die Turingmaschine alle geratenen Elemente verrechnet, vergleicht sie den Wert auf ihrem Arbeitsband mit dem Zielwert und akzeptiert bei Gleichheit. □

Unsere Untersuchungen zeigen, dass das Problem Q-UNARY-SUBSETSUM einige spannende Eigenschaften aufweist: So wächst die Kodierung des Wertes – den sich eine Turingmaschine merken muss – nicht zwangsweise, da sich die Brüche eventuell kürzen lassen. Andererseits ist der Platz, der für die Berechnung der zu speichernden Summe nötig ist, nicht logarithmisch beschränkt, denn das Ausmultiplizieren der Brüche kann Werte mit einer sehr großen Codierungslänge liefern. Das Problem Q-UNARY-SUBSETSUM hat sich somit insgesamt als interessant herrausgestellt und sollte daher unbedingt das Objekt weiterer Forschung sein.

5 Zusammenfassung und Ausblick

Wir haben in dieser Arbeit eine Vielzahl an Varianten von UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM zusammengetragen und komplexitätstheoretisch untersucht. Dabei bietet die Arbeit einen Überblick über eine Beweistechnik für die TC^0-Vollständigkeit von UNARY-SUBSETSUM und einigen seiner Varianten. Des Weiteren haben wir den sehr technischen und schwer verständlichen Beweis von Monien für die NL-Vollständigkeit des Problems PARTIAL-UNARY-SUBSETSUM für ein besseres Verständnis analysiert und überarbeitet. Mit der NL-Vollständigkeit von PARTIAL-UNARY-SUBSETSUM konnten wir anschließend einige weitere Varianten von UNARY-SUBSETSUM, wie zum Beispiel das stärker beschränkte TUBE-UNARY-SUBSETSUM, ebenfalls als NL-vollständig klassifizieren. Zusammen bietet sich die Arbeit somit als Nachschlagewerk für einige natürliche Varianten von UNARY-SUBSETSUM und deren Berechnungskomplexitäten an und beschreibt außerdem die Werkzeuge, die zum Betrachten weiterer Varianten nützlich sein können.

Während unserer Untersuchungen konnten wir zeigen, dass die meisten Varianten von UNARY-SUBSETSUM vollständig für die Komplexitätsklasse TC^0 sind, während alle Varianten des Problems PARTIAL-UNARY-SUBSETSUM vollständig für die Komplexitätsklasse NL sind. Der wesentliche Unterschied zwischen den Problemen UNARY-SUBSETSUM und PARTIAL-UNARY-SUBSETSUM ist eine untere oder obere Schranke, die der bei der Berechnung gespeicherte Wert stets über- bzw. unterschreiten muss. Diese Beschränkung ist bei der Berechnung so komplex, dass wir sie unter Standardannahmen weder mit einem TC^0-Schaltkreis, noch mit einer deterministischen Logspace-Turingmaschine entscheiden können. Betrachten wir diese Erkenntnis zusammen mit der aus Abschnitt 2.3 bekannten Hierarchie

$$TC^0 \subseteq L \subseteq NL,$$

so stellt sich abschließend die offene Frage, ob eventuell auch Varianten von UNARY-SUBSETSUM existieren, die vollständig für die Komplexitätsklasse L sind und was für Eigenschaften diese besitzen. In dieser Arbeit wurde

bereits die Lokalitätseigenschaft abgeschwächt, indem nur für einige Zwischenschritte eine Eigenschaft gefordert wird; doch diese Forderung war schon zu schwach, denn das so entstandene WAYPOINT-UNARY-SUBSETSUM konnten wir als TC^0-vollständig klassifizieren. Ein Ansatz für weitere Untersuchungen bietet die in dieser Arbeit vorgestellte Version über den rationalen Zahlen, denn für diese ist es im Rahmen dieser Arbeit nicht gelungen, eine Vollständigkeitsaussage für eine Komplexitätsklasse zu formulieren. Bei dieser Variante von UNARY-SUBSETSUM besteht die Schwierigkeit nicht in einer Schranke, sondern im immensen Wachstum von unären Zahlen bei der Multiplikation.

Register der betrachteten Varianten

Name des Problems	Komplexität	Definition	Beweis
UNARY-SUBSETSUM	TC^0-vollständig	2	17
PARTIAL-UNARY-SUBSETSUM	NL-vollständig	3	20
UNARY-KNAPSACK	TC^0-vollständig	38	39
PARTIAL-UNARY-KNAPSACK	NL-vollständig	38	41
INTEGER-UNARY-SUBSETSUM	TC^0-vollständig	53	41
PARTIAL-INTEGER-UNARY-SUBSETSUM	NL-vollständig	53	56
TUBE-UNARY-SUBSETSUM	NL-vollständig	48	48
ℓ-UNARY-SUBSETSUM	TC^0-vollständig	43	43
ℓ-PARTIAL-UNARY-SUBSETSUM	NL-vollständig	43	44
ℓ-LINEAR-PROGRAMMING	TC^0-vollständig	45	45
CHOOSE-k-UNARY-SUBSETSUM	TC^0-vollständig	50	51
CHOOSE-k-PARTIAL-UNARY-SUBSETSUM	NL-vollständig	50	52
WAYPOINT-UNARY-SUBSETSUM	TC^0-vollständig	46	46
Q-UNARY-SUBSETSUM	NL-hart, in NP	56	57, 60

Literatur

[1] Bodlaender, H. L.: A linear time algorithm for finding tree-decompositions of small treewidth. In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, ACM, New York, USA, 1993, S. 226–234

[2] Courcelle, B.: Graph rewriting: An algebraic and logic approach. In: *J. van Leeuwen, editor, Handbook of Theoretical Computer Science*, Elsevier and MIT Press, 1990, S. 193–242

[3] Diestel, R.: *Graphentheorie*. 4. Springer, 2010

[4] Elberfeld, M. ; Jakoby, A. ; Tantau, T.: Logspace Versions of the Theorems of Bodleander and Courcelle. In: *Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society, 2010, S. 143–152

[5] Elberfeld, M. ; Jakoby, A. ; Tantau, T.: Algorithmic Meta Theorems for Circuit Classes of Constant and Logarithmic Depth. In: *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2012, S. 66–77

[6] Flum, J. ; Grohe, M.: *Parameterized Complexity Theory*. Springer, 2006

[7] Garey, M. R. ; Johnson, D. S.: *Computers and Intractability – A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979

[8] Immerman, N.: *Descriptive Complexity*. Springer, 1999

[9] Monien, B.: Connections between the LBA Problem and the Knapsack Problem. In: *Begriffsschrift Jenaer Frege-Konferenz*, Friedrich-Schiller-Universität Jena, 1979 (Wissenschaftliche Beiträge der Friedrich-Schiller-Universität Jena), S. 262–280

[10] Nešetřil, J. ; Mendez, P. O.: Tree-depth, subgraph coloring and homomorphism bounds. In: *European Journal of Combinatorics*, Elsevier, Netherlands, 2005, S. 193–242

[11] Odlyzko, A. M.: The rise and fall of knapsack cryptosystems. In: *Cryptology and computational number theory* Bd. 42, A.M.S, 1990, S. 75–88

[12] Papadimitriou, C. H.: *Computational Complexity*. Addison Wesley, 1993

[13] Rosser, J. B. ; Schoenfeld, L.: Approximate formulas for some functions of prime numbers. In: *Illinois Journal of Mathematics* Bd. 6, Department of Mathematics, University of Illinois, 1962, S. 62–94

[14] Ruzzo, W. L.: On Uniform Circuit Complexity. In: *Journal of Computer and System Science* Bd. 22, Elsevier, Netherland, 1981, S. 365–383

[15] Stockhusen, C.: *Anwendungen monadischer Logik zweiter Stufe auf Probleme beschränkter Baumweite und deren Platzkomplexitäten*, Universität zu Lübeck, Diplomarbeit, 2011

[16] Vollmer, H.: *Introduction to Circuit Complexity*. Springer, 1999