Algorithmen für das Technology Mapping

Alexander Zorn

Geboren am 26. Mai 1996 in Bonn

26. Juli 2018

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Stephan Held

Zweitgutachter: Prof. Dr. Dr. h.c. Bernhard Korte

Forschungsinstitut für Diskrete Mathematik

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Inhaltsverzeichnis

1 Einleitung

Die ständig komplexer werdenden Anforderungen an die Informationstechnologie verlangen nach immer leistungsfähigeren Computerchips.
Die sich hieraus ergebenden Anforderungen an die Chipentwicklung wurden bereits 1965 von Gordon Moore in "Moore's law"[Moo65] beschrieben. Hiernach ist regelmäßig eine Verdopplung der Integrationsdichte, der Anzahl von Transistoren pro Flächeneinheit erforderlich und auch bisher technisch realisierbar. Nach [KHF18] wird dies jedoch in absehbarer Zeit nicht mehr möglich sein, wodurch Lösungsansätze durch Diskrete Mathematik, die auch Thema dieser Arbeit sind, einen wichtigen Beitrag für das Chip-Design leisten.
Derzeit beträgt die Dichte an Transistoren, die zu einem integrierten Schaltkreis auf einem Chip miteinander verbunden sind, bis zu mehrere Milliarden. Diese sind so angeordnet, dass sie gemeinsam eine vorgegebene logische Funktion errechnen können. Die Aufgabe des Chip-Designs ist es, einen herstellbaren Chip zu entwerfen, der eine vorgegebene logische Funktion realisiert.

Mithilfe von aus wenigen Transistoren konstruierten Bauteilen (genannt Gates, z.B.: AND, OR, INV, OAI ...), lässt sich eine logische Funktion nachbilden. Abbildung 1 (linker Teil) zeigt dies an einem kleinen Beispiel. Die Realisierung einer solchen Funktion ist jedoch nicht eindeutig, wie Abbildung 1 zeigt. Die Größe der Menge aller möglichen Baupläne (später Circuits) für eine logische Funktion hängt maßgeblich von den auf dem Chip zur Verfügung stehenden Bauteilen sowie von dem Aufbau der Funktion, ab. Hierdurch bedingt ergeben sich eine Vielzahl von Möglichkeiten eine logische Funktion zu realisieren.

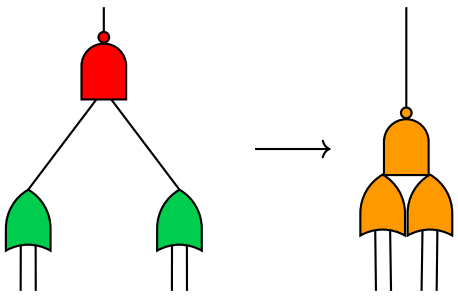


Abbildung 1: Zwei Realisierungen der logischen Funktion $\neg((w \vee x) \wedge (y \vee z))$

Jedes Bauteil hat entsprechend spezifische physikalische Eigenschaften an Größe, Geschwindigkeit (Delay) etc.. Demnach hat auch jeder Circuit entsprechende Eigenschaften.
Ziel des Technology Mappings ist es für eine gegebene logische Funktion eine Realisierung zu finden, welche eine Kostenfunktion, bestehend aus den physikalischen Eigenschaften, optimiert. Die Wahl der Lösung hat direkte Auswirkungen auf die Schnelligkeit, die Größe und den Stromverbrauch des fertigen Chips. Hierbei geht das Technology Mapping von einer bereits realisierten logischen Funktion aus und baut diese zu einer möglichst kostengünstigen Variante um.
Der optimale mögliche Umbau lässt sich bei gegebenen Bauplänen mit speziellen Eigenschaften (z.B.: eingeschränkter Größe oder Struktur) noch in kurzer Zeit finden. Die Lösung dieses Problems für allgemeine Baupläne und Kostenfunktionen ist jedoch NP-vollständig. Aus diesem Grund entwi-

ckelt die vorliegende Arbeit eine Heuristik, welche für sehr große Baupläne, bestehend aus mehreren 10.000 Bauteilen, in möglichst kurzer Zeit einen kostenoptimierten Umbau approximiert.

Bereits 1987 entwickelte Keutzer in [Keu87] einen Technology Mapping Algorithmus auf Bäumen und eine Heuristik, welche die Instanzen in Bäume zerteilt und diese optimal, bezüglich Geschwindigkeit, löst. Hieraus enwtickelte Tran in [Tra15] einen Algorithmus für Area oder Delay optimales Technology Mapping auf Bäumen.
Elbert konstruierte daraufhin 2017 in [Elb17] ein FPTAS für das Technology Mapping unter Arrivaltimeschranke oder Konvexkombination mit konstant vielen Knoten, welche mehr als eine ausgehende Kante besitzen, und einem Output. Des Weiteren wurden Matche durch Finden von geeigneten Subcircuits gesucht.
Dieses FPTAS ist für solche Circuits mit wenigen Kreisen der geeignete schnelle Algorithmus zur Optimierung mittels Technology Mapping . In [Elb17] wurde ebenfalls eine Heuristik auf Basis des FPTAS entwickelt, welche mit beliebig vielen Knoten ,welche mehrere ausgehende Kanten besitzen, in kurzer Zeit rechnen kann.

Aufbauend auf diesen Arbeiten wird, ausgehend von einem Kern-Algorithmus in Kapitel 2 auf eingeschränkten Instanzen, in Kapitel 3 die polynomielle Laufzeit des FPTAS bewiesen und die Heuristik aus [Elb17] beschrieben. Darüber hinaus wird die Match-Suche verallgemeinert, um mit einer nahezu beliebigen Menge von Gates arbeiten zu können.
Daraufhin wird in Kapitel 4 diese Heuristik so erweitert, dass Anforderungen an die späteste Ankunftszeit von Signalen an den Knoten berücksichtigt werden können und mit mehreren Outptus gerechnet werden kann.
In [DHHS18] wurde 2018 wird ein Algorithmus zur Lösung des Gate Sizing Problems mittels dem Resource Sharing Problem entwickelt. Das Schema dieses Algorithmus wird in dieser Arbeit auf einen Algorithmus zur Lösung des Technology Mapping Problems mithilfe des Resource Sharing übertragen. Dadurch wird entwickelte Heuristik Teil einer allgemeinen Logik-Optimierung von Circuits.
Abschließend wird in Kapitel 6 die Implementierung der vorgestellten Heuristik ausführlich analysiert.

Ich möchte mich an dieser Stelle ganz herzlich bei Prof. Dr. Dr. h.c. Bernhard Korte, Professor Dr. Stephan Held, Professor Dr. Jens Vygen und Professor Dr. Stefan Hougardy für die hilfsbereite und ausgezeichnete Unterstützung bei dieser Arbeit im Forschungsinstitut für Diskrete Mathematik in Bonn bedanken.
Dieser Dank richtet sich ebenfalls an alle Mitarbeiter des Instituts welche für fachliche Fragen und zahlreiche Korrekturlesungen mir jederzeit zur Seite standen.
Besonders hervorheben möchte ich hierbei Professor Dr. Stephan Held und Frau Anna Hermann für die unermüdliche Hilfe bei der Problembeseitigung im Zuge der Implementierung der vorzustellenden Heuristik.

2 Terminologie und Kern-Algorithmus

2.1 Grundlegende Definitionen

Es folgen einige grundlegende Definitionen, die für die Beschreibung des Problems erforderlich sind.

Definition 2.1. Boolesche Variable und Funktion:
Eine boolesche Variable ist eine Variable mit Werten in $\{0,1\}$. Sei $n, m \in \mathbb{N}$. Eine boolesche Funktion ist eine Funktion $f : \{0,1\}^n \to \{0,1\}^m$ mit n Inputs und m Outputs.

Definition 2.2. Gate und Library:
Ein Gate g mit Eingangsgrad (arity) $n \in \mathbb{N}$ ist ein Tripel $(f_g, d_g, area_g)$. Hierbei sind $d_g, area_g \in \mathbb{R}_{\geq 0}$. Des Weiteren ist f_g eine boolesche Funktion mit $f_g : \{0,1\}^n \to \{0,1\}$. Die Eingänge und der Ausgang eines Gates werden auch Pins genannt.
Eine Library L ist eine Menge von Gates und sei:

$$fanin_{max} := max\{arity(g) | g \in L\}$$

.

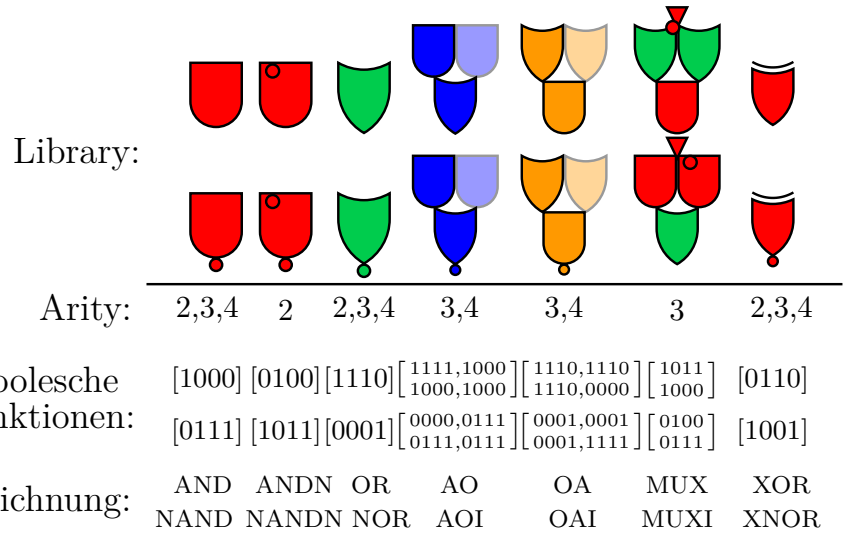


Abbildung 2: Beispiel einer Library

In der Praxis treten Gates mit gleicher Funktionalität sehr häufig auf. Diese unterscheiden sich hinsichtlich Größe, Geschwindigkeit und Stromverbrauch etc.. Die Entscheidung, welche Variante des Gates implementiert wird, wird mit einem Gate-Sizing Algorithmus getroffen. Eine ausführliche Beschreibung und effiziente Lösung dieses Problems findet sich in [HH15]. In dieser Arbeit wird davon ausgegangen, dass in einer Library keine logische Funktion durch zwei verschiedene Elemente realisiert wird.
Abbildung 2 gibt einen Überblick über eine praktisch genutzte Library. Jedes dieser Gates liegt in diesem Beispiel auch in einer invertierten Version vor (zweite Reihe der Library). Des Weiteren ist es gewöhnlich so, dass von

einem Gatetyp mehrere Varianten bezüglich der Anzahl seiner Inputs vorhanden sind. Die Zeile mit dem Titel Arity gibt einen Überblick darüber, welche Variationen auf Chips üblich sind.
In der entsprechenden Zeile der booleschen Funktionen der Abbildung ist für jedes Gate (einmal nicht invertiert und einmal invertiert) die realisierte Funktion, für 2, 3 oder 4 Inputs, abgebildet. Dies geschieht in folgendem Format: Der Ausdruck $[1, ..., 2^{arity(g)}]$ gibt für ein Gate g an der Stelle $i \in \{1, ..., 2^{arity(g)}\}$ die Wahrheitsbelegung des Outputs bei Inputbelegung i_2 (i dargestellt in Binärschreibweise) an. Es ist üblich an den Namen eines Gates die Arity zu knüpfen. Beispiel: AND2.

Die $area_g$ gibt die Größe des physikalischen Bauteils an und d_g beschreibt die Zeit, die ein Signal braucht, um von den Inputs des Gates zu seinem Output zu gelangen. Dieser Wert lässt sich noch weiter differenzieren, indem man $d_g \in \mathbb{R}^{arity(g)}$ wählt und somit Zeiten für jeden der Inputs angegeben werden können (siehe Kapitel 4.4). Wenn die Signale der Inputs nicht zur selben Zeit ankommen, wird, falls nichts anderes angegeben, gewartet, bis das letzte Signal das Gate erreicht.

Definition 2.3. Circuit:
Ein Circuit ist ein gerichteter kreisfreier Graph (DAG directed acyclic graph) mit der Eigenschaft: Jeder Knoten gehört zu einer der aufgelisteten Kategorien:

- **Inputs** mit Eingangsgrad Null.
- **Gates** mit mindestens einer eingehenden und mindestens einer ausgehenden Kante. Diese Knoten entsprechen den Gates (Definition 2.2). Darüber hinaus gilt: An jeder eingehenden Kante kann ein Inverter liegen.
- **Outputs** mit genau einer eingehenden und keiner ausgehenden Kante.

Ein Gate mit mehr als einer ausgehenden Kante wird auch Multifanoutgate genannt. Ein Circuit realisiert durch Verschachtelung der booleschen Funktionen seiner Gates ebenfalls eine boolesche Funktion. Zwei Circuits heißen äquivalent, wenn sie die gleiche boolesche Funktion realisieren.

In den Abbildungen dieser Arbeit werden die Outputs eines Circuits nicht mit angegeben, da eindeutig ist, welches Gate Vorgänger eines Outputs ist. Diese Knoten sind gemeint, wenn von den Outputs des Graphen gesprochen wird.
Es ist möglich, dass das Signal eines Gates sowohl direkt in einen Output-Knoten geleitet als auch noch weiter verarbeitet wird. Weitere Informationen über den Umgang mit mehreren Outputs befinden sich in Kapitel 4.1.
In einem Circuit C lassen sich Teilgraphen durch ein Gate der Library austauschen, vorausgesetzt die logische Funktion von C ändert sich nicht:

Definition 2.4. Match und Kandidat:
Sei g ein Gate in einem Circuit C. Ein (invertiertes) Match m auf g ist ein Tupel (p_m, X_m, f_m, inv_m), welches Folgendes enthält:

- Ein Gate p_m der Library
- Eine Menge X_m von Knoten aus dem Circuit und eine Bijektion $f : X_m \to inputs(p_m)$
- Eine Funktion $inv_m : inputs(p_m) \to \{not\_inv, inv\}$,

sodass der Circuit C', welcher durch den Austausch des Subcircuits von X_m bis g durch das Match (mit den durch inv_m definierten Invertern an den Inputs) entsteht, äquivalent zu C ist. Ein invertiertes Match auf g ist ein Match auf g mit einem Inverter an jedem seiner Outputs.
Ein Tupel, welches die Äquivalenzbedingung evtl. nicht erfüllt, wird potenzielles Match genannt.
Ein (invertierter) Kandidat auf g besteht aus einem (invertierten) Match auf g und einem Kandidaten für jeden Input Knoten von g (welcher kein Input von C ist).

Definition 2.5. Circuit-Kandidat:
Sei C ein Circuit mit Output-Knoten-Menge O. Ein Circuit-Kandidat K von C ist eine Menge von Kandidaten, sodass für jeden Output aus C genau ein Kandidat in O vorhanden ist und je zwei Kandidaten aus O an überschneidenden Knoten dasselbe Match bilden.

Abbildung 3 visualisiert die vorherigen Definitionen.

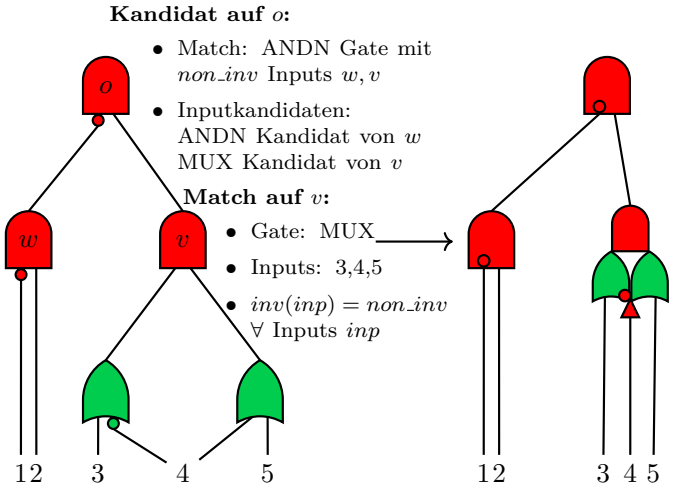


Abbildung 3: Beispiel eines Kandidaten und eines Matches

Ein Circuit-Kandidat auf einem Circuit C beschreibt eine mögliche Realisierung der C zugrunde liegenden Funktion. Wie bereits in der Einleitung dargestellt, gilt es nun den, bezüglich eines Maßes, besten Circuit-Kandidaten auf C zu finden.

Wie bereits in der Gate-Definition erwähnt, braucht ein Signal eine gewisse Zeit (Delay) um das Gate zu passieren. Dies gilt ebenfalls für jede Kante eines Circuits. Abhängig von dem Ort der Endknoten und der Lage auf dem Chip besitzt jede Kante einen Delay-Wert.
Des Weiteren ist für die Inputs des Circuits eine Arrivaltime (AT) gegeben, die angibt, zu welchem Zeitpunkt das Signal an diesem Input ankommt. Von den Inputs ausgehend lässt sich die Arrivaltime mithilfe der Delay-Werte durch den gesamten Circuit propagieren und entsprechende Werte für die Outputs angeben. Hierbei gilt, dass die Arrivaltime eines Knotens die Ankunftszeit des letzten Signals der Inputs am Output des Knotens angibt. Dies lässt sich auch auf einen Kandidaten übertragen:

Definition 2.6. Area und Delay eines Kandidaten:
Sei C ein Circuit und K ein Circuit-Kandidat auf C, dann gilt:

- $area(K) = \sum_{g \in gates(K)} (area_g + \sum_{i \in inputs(g)} \mathbb{1}_{inv_g(i)} area_{inv})$
- $AT(K) = \max_{k \in can(K)} \{ \max_{i \in inputs(k)} \{ d_{gate(k)} + \mathbb{1}_{inv_g(i)} d_{inv} + AT(inp\_can(k,i)) + d_{w(k,i)} \} \}$

Wobei $can(K)$ die Menge der Kandidaten von K ist, $area_{inv}$ und d_{inv} sind Größe und Delay eines Inverters. Darüber hinaus sind $inputs(k)$ die Input-Knoten des Output-Knoten des Kandidaten k. Des Weiteren ist $d_{w(k,i)}$ das Delay der Kante zwischen den Knoten k und i. Hierbei gibt $inp\_can(k,i)$ den Kandidaten des i'ten Inputs von k zurück. Die $gates(K)$ entsprechen allen durch den Kandidaten K festgelegten Gate-Knoten.

2.2 Kern-Algorithmus

Es folgt ein grundlegender Algorithmus, der auf nachfolgend dargestellten eingeschränkten Circuits arbeitet, jedoch im weiteren Verlauf dieser Arbeit zu einer Heuristik für allgemeine sehr große Circuits erweitert wird.

(einfaches) Technology Mapping	
Instanz:	Circuit C ohne Multifanoutknoten, mit eindeutigem Output o, Library L mit beschränktem $fanin_{max}$
Aufgabe:	Finde einen Kandidaten K auf o, welcher die Arrivaltime/Area minimiert.

Algorithmus : (einfaches) Technology Mapping

Input : Circuit C kreisfrei mit Output o, Library L

```
1 bester_kandidat[] ← ∅
2 bester_inv_kandidat[]← ∅
3 foreach Knoten v ∈ V(C) in topologischer Reihenfolge do
4 |   berechne_alle_(invertierten)_Matche(v)
5 |   foreach Match m auf v do
6 |   |   k, k_inv ←bester_(inv)_Kandidat(m, v)
7 |   |   Update_best_(inv)_Kandidat(k, k_inv)
8 Implementiere C entsprechend bester_kandidat[o]
```

Dieser Algorithmus stammt aus [Elb17] und wurde 1987 in [Keu87] entwickelt.
Ein Technology Mapping (TM) Algorithmus liefert in der Regel nicht die bestmögliche Realisierung der C zugrunde liegenden logischen Funktion.

Dies veranschaulicht Abbildung 4, welche die Realisierung einer konstanten Funktion darstellt, die keine Gates benötigt, was jedoch mit den bisher eingeführten Möglichkeiten des Technology Mappings nicht realisierbar ist.

Korollar 2.7. *Das einfache Technology Mapping liefert bezüglich der vorgestellten Möglichkeiten des Matchens und der Kandidatenbildung den bestmöglichen äquivalenten Circuit.*

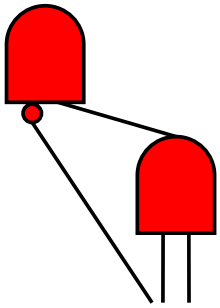


Abbildung 4: Ein Circuit, dessen boolesche Funktion $f = 0$ ist

Beweis: Der Algorithmus geht in topologischer Reihenfolge durch die Knoten v des Graphen und berechnet alle Matche auf v. Diese werden dann zu einem Kandidaten ergänzt. Ohne Multifanoutknoten überschneiden sich diese nicht. Für jeden Knoten und jedes Match gibt es nur einen Kandidaten zur Auswahl, da für die Inputs des Matches jeweils nur ein Kandidat gespeichert wurde. An jedem Knoten wird nur das Match mit dem dazugehörigen Kandidaten gespeichert, welches die Kosten optimiert. Es bleibt zu zeigen, dass, angenommen, dass für alle Knoten mit kleinerem topologischen Rang als $rang(v)$ der bestmögliche Kandidat bereits gespeichert ist, dann auch für v der beste Kandidat k gespeichert wird. Angenommen, es gibt einen besseren Kandidaten k' als k, welcher von dem Algorithmus gespeichert wurde. Sei k'' der Kandidat, welcher dasselbe Match wie k' benutzt und die besten Input-Kandidaten. Da k'' die besten Input-Kandidaten benutzt, ist er mindestens so schnell (bzw. klein) wie k'. Jedoch ist k ebenfalls mindestens so kostengünstig wie k'' (andernfalls hätte der Algorithmus k'' k vorgezogen). Dies ist ein Widerspruch zur Annahme. □

Korollar 2.8. *Der Algorithmus für das (einfache) Technology Mapping besitzt $\mathcal{O}(|V(C)|^3|L|^2)$-Laufzeit.*

Beweis: Schritt 1 und 2 besitzen Laufzeit $\mathcal{O}(1)$. Schritt 4 lässt sich aufgrund von einem beschränkten $fanin_{max}$ und ohne Multifanoutgates in $\mathcal{O}(|V(C)|^2|L|)$ errechnen. Der Beweis dieser Aussage befindet sich in Kapitel 3.5.
Der Schritt 6 ist schnell implementierbar, da für jeden der max $fanin_{max}$ Inputs der beste Kandidat bereits errechnet wurde und somit nur verlinkt werden muss. Ein invertiertes Match wird nur gebraucht, wenn der korrespondierende Input des darüber liegenden Gates invertiert ist. Schritt 6 lässt sich somit in $\mathcal{O}(fanin_{max})$ realisieren. Die Schritte 3 und 5 sind zwei verschachtelte Schleifen mit $|V(C)|$ und $\mathcal{O}(|V(C)|^2|L|)$ Durchläufen.
Daraus folgt eine Laufzeit von $\mathcal{O}(|V(C)|^3|L|)$. □

3 Entwicklung von FPTAS und Heuristik

In diesem Kapitel wird ein Approximationsalgorithmus (ein FPTAS) für allgemeinere Instanzen und Zielfunktionen des Technology Mappings auf Basis von [Elb17] vorgestellt. Da auch dieser, um eine polynomielle Laufzeit zu erreichen, noch einige Einschränkungen an gegebene Circuits hat, die für reale Instanzen eines Chips nicht gelten und da sich herausstellen wird, dass, nach [KR89], das Technology Mapping bereits mit einer Konvexkombination als Kostenfunktion NP-vollständig ist, wird auf Basis des Approximationsalgorithmus eine Heuristik entworfen.

3.1 Tradeoffprobleme

Der in Kapitel 2.2 vorgestellte Algorithmus ist in der Lage, den bestmöglichen Umbau eines eingeschränkten Circuits bezüglich Area oder Delay zu errechnen.
Es existiert ein Tradeoff zwischen Area und Delay. Dies hat zur Folge, dass ein möglichst kleiner Circuit in der Regel sehr langsam ist und bei einem schnellen Circuit ein großer Platzverbrauch zu erwarten ist. Bei der Lösungsentwicklung mittels Technology Mapping ist jedoch weder ein sehr langsamer noch ein besonders großer Circuit akzeptabel.
Daraus folgt die Suche nach einem Algorithmus, welcher in der Lage ist, Circuits bezüglich einer Konvexkombination oder einer Schranke zu verbessern. Hierbei ergeben sich die beiden folgenden Optimierungsprobleme:

Technology Mapping mit Konvexkombination	
Instanz:	Circuit C, mit einem Output, Library L mit beschränktem $fanin_{max}$, $\|L\|$ beschränkt und Tradeoffparameter $\lambda \in [0,1]$.
Aufgabe:	Finde einen Circuit-Kandidaten K auf C, welcher $\lambda AT(K) + (1-\lambda) area(K)$ minimiert.

Technology Mapping mit Arrivaltimeschranke	
Instanz:	Circuit C, mit einem Output, Library L mit beschränktem $fanin_{max}$, $\|L\|$ beschränkt und Arrivaltimeschranke A_{max}.
Aufgabe:	Finde den kleinsten Circuit-Kandidaten K auf C, für den $AT(K) \leq A_{max}$ gilt, oder entscheide, dass für jeden Circuit-Kandidaten K bereits $AT(K) > A_{max}$ gilt.

In Kapitel 4.1 werden diese Problemstellungen auf Circuits mit mehreren Outputs erweitert. Beide Probleme lassen sich mit dem noch vorzustellenden FPTAS lösen. Eine Beschreibung, wie sich das Technology Mapping mit Arrivaltimeschranke mit dem FPTAS lösen lässt, findet sich in [Elb17].
Da im weiteren Verlauf dieser Arbeit die Arrivaltime nicht der einzige zu beachtende Faktor bei der Geschwindigkeitsoptimierung ist, wird im Folgenden nur noch die Konvexkombination behandelt. Diese Erweiterungen lassen sich jedoch auch mit in das Technology Mapping mit Arrivaltimeschranke integrieren.

Bei der Implementierung eines Algorithmus für die beiden Optimierungsprobleme ergibt sich folgende Herausforderung:
Der kostengünstigste Kandidat an einem Knoten v benutzt nicht unbedingt die kostenünstigsten Kandidanten der Inputs des Matches von v. Es ist nämlich möglich, dass einer der Inputs sehr AT kritisch ist und der Subcircuit unter dem anderen Input sehr viel Area benötigt. Dadurch ist es möglich, dass der kostengünstigste Kandidat auf v dem einen Input einen sehr schnellen und dem anderen Input eine sehr kleinen Kandidaten zuweist. Speichert man an jedem Knoten nur den kostenminimierenden noch vorhandenen Kandidaten, bleibt keine Garantie, dass der kostengünstigste Circuit-Kandidat am Output noch vorhanden ist. Die Kosten eines Kandidaten k sind aus diesem Grund nicht $\lambda AT(k) + (1-\lambda)area(k)$, sondern das Tupel $(AT(k), area(k))$. Es gibt jedoch eine Klasse von Kandidaten, welche nicht gespeichert werden muss:

Definition 3.1. Dominierte Kandidaten:
Seien k_1, k_2 Kandidaten desselben Knotens. Dann wird k_2 von k_1 dominiert wenn mindestens eine der folgenden Bedingungen erfüllt ist:

- $AT(k_1) < AT(k_2)$ und $area(k_1) \leq area(k_2)$
- $AT(k_1) \leq AT(k_2)$ und $area(k_1) < area(k_2)$.

Eine optimale Lösung des Technology Mappings verwendet nur nicht dominierte Kandidaten, denn sonst ließe sich durch Ersetzen eines dominierten durch einen nicht dominierten Kandidaten eine bessere Lösung erzielen. Daraus folgt, dass nur die nicht dominierten Kandidaten während der Ausführung des Algorithmus gespeichert werden müssen. Die Menge der noch verbleibenden Kandidaten lässt sich in einer sogenannten Tradeoff-Kurve speichern (s. Abb. 5), die jeden Kandidaten zweidimensional anhand seiner Kosten erfasst.

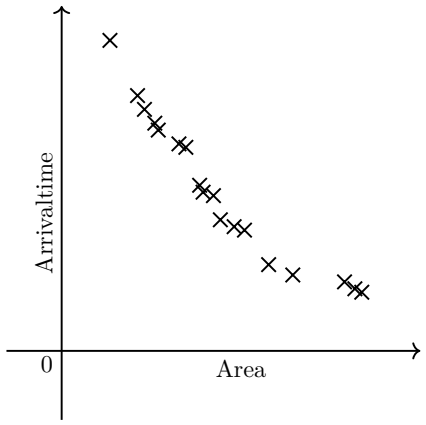


Abbildung 5: Ordnen der Kandidaten in einer Tradeoffkurve

Die beiden vorgestellten Probleme sind NP-vollständig. Ein Beweis dieser Aussage und genauere Informationen zur Einordnung der Schwere dieser Probleme finden sich in [KR89] und [Elb17].
Daraus folgt, dass sich ab diesem Punkt wahrscheinlich kein polynomieller Algorithmus für das Technology Mapping finden lässt der, hinsichtlich der vorgestellten Operationen den kostengünstigsten Circuit-Kandidaten liefert. Da bei zwei nicht dominierten Kandidaten nicht eindeutig ist, welcher der bessere ist, wird eine Vielzahl von Kandidaten an jedem Knoten gespeichert, was zu einem exponentiell großen Speicherbedarf führt.

3.2 Multifanoutknoten

Der beschriebene Kern-Algorithmus arbeitet nur auf Circuits, in denen keine Multifanoutknoten existieren. Diese kommen auf einem realen Chip jedoch zu ca. 30% vor. Eine Analyse über die Auswirkungen, die sich aus der Anzahl der Multifanoutknoten auf die Laufzeit ergeben, befindet sich in Kapitel 6.
Es ist möglich, einen Circuit in kleinere Subcircuits zu unterteilen, die solche Multifanoutknoten nicht beinhalten. Diese Subcircuits werden einzeln mit dem Algorithmus (sehr schnell) optimiert und daraufhin zu einem C äquivalenten Circuit C' zusammengesetzt. Diese Vorgehensweise findet sich ausführlich in [Keu87] wieder.

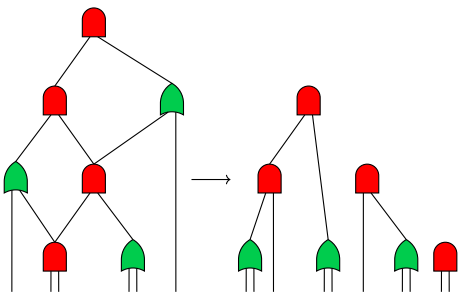


Abbildung 6: Unterteilen eines Circuit in Multifanoutfreie Subcircuits

Abbildung 6 stellt diesen Ansatz einer Heuristik dar.
Der Anteil an Multifanoutknoten ist auf den mir vorliegenden Chips so groß, dass eine Vielzahl sehr kleiner Subcircuits entstehen, woraus folgt, dass die Möglichkeiten des Technology Mappings sehr eingeschränkt werden. Aus diesem Grund werde ich diesen Weg einer Heuristik nicht weiter verfolgen.
Bei der Implementierung von Multifanoutknoten besteht die größte Herausforderung darin, dass bei der Konstruktion des äquivalenten Circuits die eingebauten Kandidaten aller Nachfolger eines Multifanoutknoten v an v übereinstimmen müssen. Daraus folgt, dass bei der Wahl eines Kandidaten für einen Knoten w die Input-Kandidaten von w abhängig voneinander gewählt werden müssen.

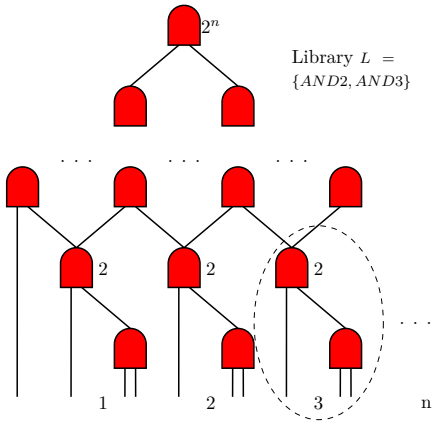


Abbildung 7: Exponentiell viele Kandidaten bereits bei sehr eingeschränkter Library

Abbildung 7 zeigt zudem eine weitere Herausforderung für die Implementierung auf. Die Anzahl der zu speichernden Kandidaten kann exponentiell bezüglich $|V(C)|$ sein. Zum Beispiel liegen am Output des Circuits dieser Abbildung mindestens 2^n verschiedene Kandidaten vor, da am Output des gestrichelten Subcircuits 2 Kandidaten vorhanden sind und dieser Subcircuit n mal in dem Graphen vorkommt. Die Anzahl der Knoten in dem Graphen ist polynomiell in n.
Zur Lösung der ersten Herausforderung helfen die folgenden Definitionen:

Definition 3.2. Cone eines Knoten:
Sei C ein Circuit und v ein Knoten von C, dann sei die Cone von v:

$$cone(v) := C[V \cup \{v\}], V = \{w \in V(C) : \exists \text{ w-v-Weg in } C\}.$$

Die durch die $cone(v)$ berechnete logische Funktion wird die bis v berechnete Funktion genannt.

Definition 3.3. Offene Knoten:
Sei C ein Circuit und $v, w \in V(C)$, dann heißt o offener Knoten von v, wenn Folgendes gilt:

- $o \in cone(v) \backslash \{v\}$
- $|\delta^+(o)| \geq 2$
- $\exists w \in V(C) \backslash cone(v) : \exists$ o-w-Weg in C ohne v.

Somit ist die Menge der offenen Knoten eines Circuit Knotens v die Menge aller Multifanoutknoten o, von welchen aus man sowohl v als auch einen Knoten außerhalb der Cone von v erreichen kann. In dieser Menge ist v selber nicht enthalten. Die offenen Knoten von v sind gerade die Multifanoutknoten, die durch die Kandidaten eines Knotens außerhalb von $cone(v)$ verändert werden können. Alle Kandidaten von Knoten mit Ausgangsgrad 1 und dieser Eigenschaft sind durch den Nachfolger-Kandidaten, der auch zu einem offenen Knoten gehören muss, bereits eindeutig definiert. Abbildung 8 visualisiert die vorangegangenen Definitionen. Hierbei entsprechen die mit o markierten Knoten den offenen Knoten der $cone(v)$.

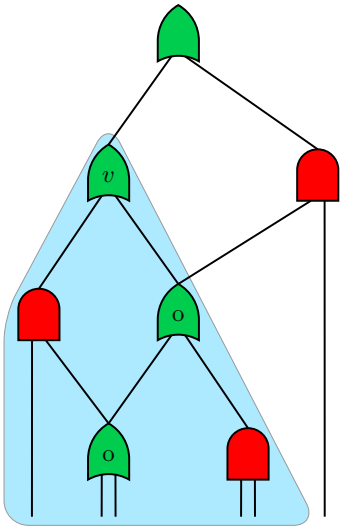


Abbildung 8: Visualisierung der Definitionen 3.2 und 3.3

Definition 3.4. Klasse eines Kandidaten:
Sei k ein Kandidat auf einem Knoten v und O die Menge der offenen Knoten von v. Die Klasse $class(k)$ ist eine Abbildung, welche jedem Element $w \in O$ den durch k festgelegten Kandidaten auf w zuordnet.

Hiernach lassen sich zwei Kandidaten k_1, k_2 eines Knotens v mit $class(k_1) \neq class(k_2)$ nicht miteinander vergleichen, auch nicht, wenn k_2 von k_1 dominiert wird, denn es ist möglich, dass dies zwar an der Stelle v gilt, jedoch nicht an allen offenen Knoten von v. Daraus folgt, würde man k_2 löschen, so löscht man evtl. den besten Kandidaten des Outputs von C.
Um daher mit Multifanoutknoten arbeiten zu können, werden für jeden Knoten v und jede Klasse von v alle nicht dominierten Kandidaten gespeichert. Dann ist der noch verbleibende beste Kandidat des Outputs die beste Lösung. Dies führt jedoch zu einem nur exponentiell beschränkten Speicheraufwand. Zur Speicherung der Kandidaten wird für jede Klasse eines Knotens eine Tradeoff-Kurve angelegt.

3.2.1 Klonen

Die Input-Kandidaten eines Kandidaten müssen aus nachfolgend dargestelltem Grund an deren offenen Knoten übereinstimmen:
Angenommen, dies wäre nicht der Fall, dann würden offene Knoten evtl. mehrere Male mit verschiedenen Kandidaten gebaut. Dieser Vorgang wird auch Klonen genannt. Dies kann von Vorteil sein, wenn zum Beispiel ein offener Knoten Teil eines sowohl sehr Delay- als auch sehr Area-kritischen Gebietes ist. Dann würde einmal ein schneller und einmal ein sehr kleiner Kandidat realisiert. Dies führt in der Regel jedoch zu einem deutlich erhöhten Platzverbrauch und es werden mehr Kanten gebraucht, was höhere Routing Kosten mit sich bringt, welche im Technology Mapping jedoch nicht berücksichtigt werden. Um einen übermäßigen Anstieg nicht beachteter Kosten zu verhindern, ist das Klonen in den vorgestellten Algorithmen nicht erlaubt und wird, durch die Sortierung der Input-Kandidaten eines Kandidaten in Klassen, verhindert.

3.3 Zu lange Kanten

Abbildung 9 veranschaulicht eine häufig auftretende Situation. Es handelt sich um das Matchen über eine auf dem Chip sehr lange Kante. Dadurch verbessert sich evtl. die Größe des Circuits, jedoch sind nach dem Match nun zwei sehr lange Kanten auf dem Chip vorhanden, was einen großen Routing-Aufwand und weitere Kosten mit sich bringt und somit eine zu vermeidende Situation ist. Diese Kanten bezeichnet man als konstant.
Bei der Bildung eines Matches wird darauf geachtet über keine konstante Kante zu matchen. Durch das Hinzufügen der zu langen Kanten zu den konstanten Kanten kann keine optimale Lösung mehr im Kern-Algorithmus garantiert werden, denn es ist möglich, aber unwahrscheinlich, dass die zusätzlichen Routing-Kosten vernachlässigbar sind und ein Match über diese Kante für eine optimale Lösung notwendig ist.

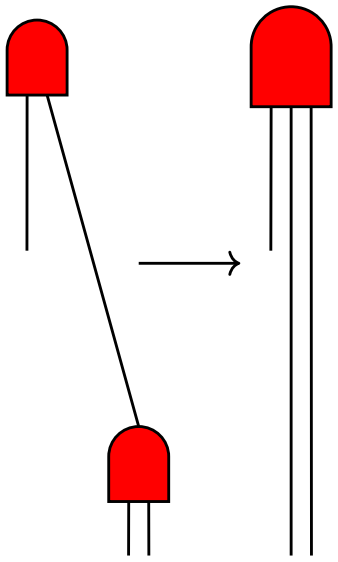


Abbildung 9: Visualisierung eines Matches über eine lange Kante

Alternativ ließe sich auch die Netzlänge eines Chips den Optimierungskriterien des Technology Mapping hinzufügen um diese zusätzlichen Kosten automatisch zu verhindern. Es bleibt jedoch offen dies zu implementieren.

3.4 Teilweise überflüssige Subcircuits

Abbildung 4 zeigt, dass nicht unbedingt alle Inputs eines Circuits relevant sind für die Outputs. Zur genaueren Einordnung folgt eine Definition:

Definition 3.5. Vollständig überflüssiger und teilweise überflüssiger Circuit: Sei C ein Circuit mit logischer Funktion $f : \{0,1\}^n \to \{0,1\}^m$.
C wird vollständig überflüssig genannt, wenn gilt:

$$\exists y \in \{0,1\}^m : \forall x \in \{0,1\}^n : f(x) = y$$

C wird teilweise überflüssig genannt, wenn es einen Input i gibt, sodass für keine Belegung x der anderen Inputs gilt: Das Bild von f unter x ist durch die Wahl der Belegung von i veränderbar.

Die Berücksichtigung von vollständig überflüssigen Subcircuits würde bedeuten, dass Teile des Circuits entfernt werden und die dann freien Pins der nachfolgenden Knoten an permanenten Strom gelegt oder mit der Erdung des Chips verbunden werden müssen. Dies lässt sich jedoch weiter optimieren. Da die Information an den nachfolgenden Gates vorhersagbar ist, muss sie auch nicht verarbeitet werden. Daraus folgt eine hohe Einsparung von Kosten. Es bedingt jedoch ebenfalls einen großen Aufwand zur Implementierung in die aktuelle Architektur des Technology Mapping Algorithmus, da für Gate mit einem solchen vorhersagbaren Pin ein Ersatz-Subcircuit gefunden werden muss, welcher diesen Pin nicht benutzt und die logische Funktion abhängig der anderen Pins realisiert. Der Technology Mapping Algorithmus muss nun wiederholt angewendet werden, da dieser Subcircuit evtl. nicht mehr zu einer guten Lösung führt. In der Praxis ist das Vorkommen von vollständig überflüssigen Subcircuits verschwindend gering. Von daher werden die vollständig überflüssigen Subcircuits in dieser Arbeit nicht weiter behandelt und werden in der Implementierung der noch folgenden Algorithmen nicht weiter berücksichtigt.
Im Gegensatz dazu kommen die teilweise überflüssigen Circuits durchaus vor. Bei der Konstruktion eines Chips geschieht dies durch das Zusammensetzen unterschiedlicher Circuits.

In den meisten Fällen werden die teilweise überflüssigen Circuits automatisch bei der Suche der Matche gefunden, da die irrelevanten Inputs nicht mehr unter den Inputs des Match auftauchen und somit beim Bau des äquivalenten Graphen verschwinden. Dies veranschaulicht Abbildung 10.

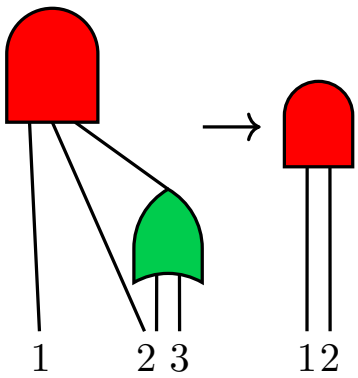


Abbildung 10: Nur Input 1 und 2 sind die relevanten Inputs

Es gibt dabei jedoch noch eine Besonderheit. Wenn alle Inputs bis auf einen irrelevant sind, ist das resultierende Gate des Matches entweder ein Inverter(INV) oder ein Buffer(BUFF). Ersteres lässt sich als Input-Invertierung des darüberliegenden Input-Pins speichern. Ein Buffer ist jedoch nicht unbedingt in der Library für das Technology Mapping vorhanden und kann vermieden werden, indem kein Buffer, sondern nur die Kanten vom Input des Buffers zu seinen Outputs gebaut werden. Dies verhindert den Einbau eines nicht nötigen Gates. Da $fanout_{max}$ und $fanin_{max}$ im weiteren Verlauf dieser Arbeit beschränkt werden, lässt sich dies in linearer Laufzeit implementieren. Diese zusätzliche Bearbeitung läuft dann in jedem

folgenden Algorithmus im Hintergrund und findet keine Erwähnung mehr. Obwohl nun das Gegenbeispiel aus Kapitel 2.2 nicht mehr gültig ist, lässt sich mit den Möglichkeiten des Technology Mappings in der Regel keine optimale Realisierung der des Circuits zugrunde liegenden logischen Funktion finden.

3.5 Polynomielle Match-Suche

In Circuits mit beliebig vielen Multifanoutknoten steigt die Anzahl der möglichen Matche eines Knotens sehr stark an, was eine direkte Auswirkung auf die Größe der Menge der möglichen Kandidaten hat.
Es folgt eine weitere Betrachtung von Matches und daraus ableitend ein Algorithmus, der auf Instanzen von real existierenden Chips fast alle möglichen Matche findet und in polynomieller Zeit implementierbar ist. Der Algorithmus zur Suche von Matchen wurde im Zuge dieser Arbeit weiterentwickelt. In [Elb17] wurden Matche mittels einer Suche geeigneter Subcircuits gefunden, was sich durch eine kurze Laufzeit auszeichnet. Jedoch müssen alle möglichen, einem Match zugrunde liegenden, Subcircuits bei der Implementierung bekannt sein. Da eine Library mit der Zeit jedoch üblicherweise um neue Gates erweitert wird, müsste bei jeder Aktualisierung nachgebessert werden.
In diesem Kapitel wird aus diesem Grund ein Ansatz vorgestellt, welcher unabhängig von der Library in der Lage ist alle Matche zu finden.

3.5.1 Obere Schranke für die Anzahl Matche eines Knotens

Folgendes Lemma legt eine äquivalente Definition für Matche nahe.
Sei ein Circuit C ohne teilweise überflüssige Subcircuits gegeben:
$\bar{C}_v := (V(C[cone(v)]), \{(u,w)|(w,u) \in E(C[cone(v)])\})$.

Lemma 3.6. *Die Inputs eines Matches auf v lassen sich eindeutig zu den Kanten eines gerichteten v-Schnitts in $\bar{C}_v$ zuordnen.*

Beweis: Zu jedem Inputpin eines Matches gehört eindeutig eine Kantenmenge K aus C. Es ist möglich, dass $|K| > 1$, denn ein Match kann mehrere ausgehende Kanten eines Knotens zu einer Kante zusammenfassen. Dies veranschaulicht Abbildung 11 für ein Match eines MUX Gates. Sei I die Menge der mit den Inputs korrespondierenden Kantenmengen.
Es genügt zu zeigen, dass alle Kanten I' in I gerade die Kantenmenge eines gerichteten v-Schnitts in $\bar{C}_v$ sind.
Angenommen I' bildet keinen gerichteten v-Schnitt, dann existiert in $\bar{C}_v \backslash I$ ein Weg von einem Input von $cone(v)$ zu v, welcher keine Kante aus I benutzt.

Eine Kante dieses Weges muss jedoch aus folgenden Gründen zu einem der Inputs des Matches gehören: Das Match hat somit einen Seitenoutput, was nicht erlaubt ist, oder es existiert ein teilweise überflüssiger Subcircuit. □

Anhand eines Schnitts kann nicht eindeutig hergeleitet werden, ob mehrere Kanten eines Outputs zusammengefasst oder getrennte Input-Kanten eines Matches sind. Dies wird ebenfalls durch Abbildung 11 veranschaulicht. Ohne Multifanoutknoten ist diese Zuweisung jedoch, abgesehen von den möglichen Invertierungen der Inputs und des Outputs, eineindeutig, wenn es keine zwei Gates der Library gibt, welche die gleiche logische Funktion realisieren.

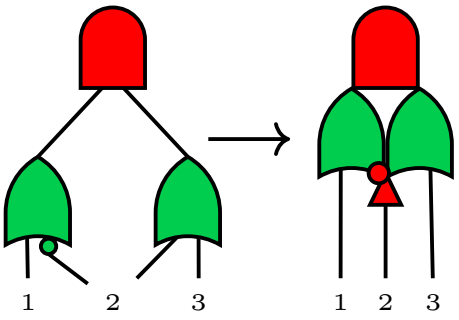


Abbildung 11: Das MUX Match verbindet zwei Kanten eines Multifanoutknoten

In einem Graphen ohne Multifanoutknoten gilt somit:
Die Menge der Matche von v ist somit gerade die Menge aller maximal $fanin_{max}$ großen v-Schnitte in $\bar{C}_v$, inklusive aller möglicher Inputinvertierungen und Outputinvertierungen, die als Subgraph die logische Funktion eines Gates der Library realisieren.

Korollar 3.7. *Sei C ein Circuit ohne teilweise überflüssige Subcircuits. Die Anzahl der Matche eines Knotens $v \in V(C)$ mit $m_v := |E(C[cone(v)]|$, wobei $C[cone(v)]$ keine Multifanoutknoten enthält ist durch $m_v^{fanin_{max}} 2^{fanin_{max}+1} fanin_{max}$ beschränkt.*

Beweis: Sei $X_v := \{E \subseteq E(C[cone(v)]) || E| \leq fanin_{max}\}$. Die Menge der maximal $fanin_{max}$ großen v-Schnitte ist in $C[cone(v)]$ durch $|X_v|$ beschränkt. Es gilt hierbei:

$$|X_v| = \sum_{i \leq fanin_{max}} \binom{m_v}{i} \leq \sum_{i \leq fanin_{max}} \frac{m_v^{fanin_{max}}}{i!} \leq m_v^{fanin_{max}} fanin_{max}.$$

Für jeden Schnitt gibt es zwei verschiedene mögliche Output-Invertierungen und maximal $2^{fanin_{max}}$ viele Möglichkeiten die Inputs zu invertieren. Jedes mögliche Match ist nun durch ein Element aus X_v und eine Wahl von Invertierungen eindeutig charakterisiert. □

Für allgemeinere Circuits lässt sich folgende Schranke angeben:

Korollar 3.8. *Sei C ein Circuit ohne teilweise überflüssige Subcircuits. Die Anzahl der Matche eines Knotens $v \in V(C)$ mit $m_v := |E(C[cone(v)]|$, ist durch*
$m_v^{fanin_{max}} fanin_{max}^2 2^{fanin_{max}+1} 2^{fanout_{max}}$ beschränkt.

Beweis: Korollar 3.7 gibt eine obere Schranke für die Anzahl aller Matche inklusive der möglichen Invertierungen für einen Circuit ohne Multifanoutknoten an. Sei $u \in x \in X_v$ (Definition siehe oben) ausgehende Kante eines Multifanoutknotens w, so gibt es bis zu $2^{fanout_{max}-1}$ Möglichkeiten, weitere Kanten von w der korrespondierenden Kantemenge von u hinzuzufügen. Diese Möglichkeit besteht für alle maximal $fanin_{max}$ Kanten von x. Durch Multiplikation mit der oberen Schranke aus Korollar 3.7 folgt die Aussage. □

In einem Circuit ohne Multifanoutknoten lässt sich die Schranke noch genauer angeben.

Korollar 3.9. *Sei C ein Circuit ohne Multifanoutknoten. Die Anzahl der Matche eines Knoten ist durch $2^{2fanin_{max}+1}fanin_{max}$ beschränkt.*

Beweis: Ausgehend von einem Knoten $v \in V(C)$ mit maximal $fanin_{max}$ Inputs gibt es $2^{fanin_{max}}$ Möglichkeiten, die an den Inputs liegenden Gates mit in das Match einzuschließen. Bis auf eine schließt jede dieser Möglichkeiten mindestens ein Gate mit ein. Da C keine Multifanoutknoten enthält, wird der Fanin des Matches um mindestens eins erhöht, denn es ist nicht möglich, Kreise zu schließen. Daraus folgt, dass dieser Schritt maximal $fanin_{max}$ mal durchführbar ist. Für jeden so berechneten Prototyp eines Matches gibt es noch $2^{fanin_{max}}$ mögliche Invertierungen der Inputs und zwei des Outputs. Daraus folgt die obige Aussage. □

Definition 3.10. Sei $\mathcal{M} := |V(C)|^{fanin_{max}} fanin_{max}^2 2^{fanin_{max}+1} 2^{fanout_{max}}$ eine Bezeichnung der oberen Schranke für die Anzahl der Matche eines Knotens in einem Graphen mit beschränktem $fanin_{max}$ und $fanout_{max}$.

3.5.2 Match-Suche in polynomieller Zeit

Der Kern-Algorithmus aus Kapitel 2.2 findet alle möglichen Matche eines beliebigen Knotens in polynomieller Zeit, da in dem Circuit keine Multifanoutknoten existieren. Aus Korollar 3.9 folgt, dass potenzielle Matche aller Knoten in $\mathcal{O}(2^{2fanin_{max}+1}fanin_{max}|V(C)|)$ errechnet werden können. Da $fanin_{max}$ als Konstante deklariert wurde, entspricht dies linearer Laufzeit. Es bleibt zu prüfen, ob ein solcher Prototyp einem Match in C entspricht, also ob die logische Funktion des Subcircuits einem Match der Library gleicht. Für jede der maximal $2^{fanin_{max}}$ möglichen Wahrheitsbelegungen der Inputs wird der Wahrheitsgehalt des Outputs errechnet. Dies ist linear in $|V(C)|$ möglich. Die dadurch errechnete Tabelle wird mit den $|L|$ Gates der Library verglichen.
Daraus folgt, dass das Finden aller Matche in $\mathcal{O}(|V(C)|^2|L|)$ erfolgt und somit polynomiell möglich ist.
Bei Circuits mit beliebig vielen Multifanoutknoten, aber beschränktem $fanout_{max}$, lassen sich nach Korollar 3.8 und gleichem Vorgehen, wie soeben beschrieben, alle Matche ebenfalls in polynomieller Zeit finden. Dabei beträgt die Laufzeit $\mathcal{O}(|V(C)|^{fanin_{max}+2}|L|)$.

3.5.3 Heuristische Match-Suche

In der Praxis wird, ähnlich der Routine ohne Multifanoutknoten, von dem Gate eines Knotens ausgehend überprüft, ob dieser oder eine mögliche Invertierung der Inputs und des Outputs einem Gate der Library entspricht. Es gibt maximal $2^{fanin_{max}}$ Möglichkeiten, das potenzielle Match mit den an den Inputs liegenden Gates zu erweitern. Für jede dieser Möglichkeiten wird ein weiteres potenzielles Match gebildet und mit jeder möglichen Invertierung der Inputs und des Outputs überprüft, ob es einem Match entspricht. Dieser Vorgang wird für jede der Möglichkeiten so lange wiederholt, bis die Anzahl der Inputs des durch das potenzielle Match beschriebenen

Subcircuits größer als $fanin_{max}$ ist. Ab diesem Punkt wird das vorliegende potenzielle Match nicht mehr erweitert.
Bei einem Circuit mit Multifanoutknoten kann hierdurch jedoch das Finden aller möglicher Matche nicht sichergestellt werden, da, sobald ein potenzielles Match einen Multifanoutknoten mit einschließt, die Zahl der Inputs sinken kann.
In der Praxis wird so jedoch die überwiegende Mehrheit der Matche gefunden. Der Verlust einiger weniger Matche ermöglicht jedoch einen enormen Laufzeitgewinn. Eine genaue Beschreibung dieser Routine findet sich in [MCB+04]. Diese Routine wird auch Boolean Matching genannt.
Mithilfe einer Boolean Matching Routine wurde diese Routine zur Suche von Matches im Zuge dieser Arbeit implementiert.

3.6 Berechnen und Filtern von Kandidaten

Die Anzahl der Kandidaten an einem Knoten ist im Allgemeinen nicht polynomiell beschränkt, wie Abbildung 7 zeigt. Dies gilt offenbar auch, wenn der Circuit C keine Multifanoutknoten hat. Dieses Problem wird durch das Filtern mit Buckets für beschränkt viele Multifanoutknoten in Kapitel 3.6.2 gelöst.

3.6.1 Schranken für Arrivaltime und Größe

Zur Vorbereitung auf das Filtern mit Buckets wird folgender Algorithmus eingeführt.
Für die Bearbeitung einer Tradeoffkurve ist es notwendig, Schranken angeben zu können, zwischen welchen sich alle Werte der Kurve befinden, damit sich nach dem Filtern (siehe Kapitel 3.6.2) die Anzahl der noch vorhandenen Kandidaten abschätzen lässt.
Durch den vorhandenen Tradeoff befindet sich die AT eines C äquivalenten Circuits zwischen der der schnellsten und der der kleinsten Implementierung von C. Dementsprechend befindet sich $Area$ von C zwischen der der kleinsten und der der schnellsten Lösung.
Eine untere Schranke lässt sich dadurch finden, indem man an jedem Knoten die schnellste nicht invertierte und die invertierte Kombination aus Match und Inputkandidaten wählt. Hierbei wird nicht darauf geachtet, dass sich die Kandidaten der Inputs in der gleichen Klasse befinden. Dies führt wahrscheinlich nicht zu einem äquivalenten Circuit, jedoch bildet die AT des Outputs eine untere Schranke für die AT eines äquivalenten Circuits. An jedem Knoten werden so genau zwei Kombinationen an Match und Input-Kandidaten gespeichert, was den darüberliegenden Knoten ermöglicht die Inputs frei invertieren zu können und keine Delay-Einbußen bezüglich nicht vorhandener invertierter Kandidaten akzeptiert werden müssen.
Dieser Algorithmus kann, wenn an jedem Knoten die jeweils kleinste Kombination gewählt wird, auch eine obere Schranke für die AT einer Lösung angeben.
Die Laufzeit dieses Algorithmus ist für beschränkten $fanin_{max}, fanout_{max}$ polynomiell, da an jedem Knoten nur zwei Kombinationen gespeichert werden und maximal $\mathcal{M}$ Matche existieren und jedes dieser Matche legt die

Input-Kandidaten implizit fest. Dieser Algorithmus wird im folgenden Untere Schranke Arrivaltime (Area) genannt.

3.6.2 Filtern mit Buckets

Die folgenden Lösung zur Reduzierung der Kandidatenmenge wurde in [Elb17] entwickelt und implementiert.
Die Kandidaten eines beliebigen Knotens sind in Tradeoffkurven, nach Klassen sortiert, gespeichert. Die Werte einer solchen Kurve lassen sich in Abschnitte (Buckets) fester Größe einteilen. Dabei lässt sich eine Kurve sowohl in Delay-Buckets der Größe δ_{delay} als auch in Area-Buckets der Größe δ_{area} unterteilen. Dies wird in Abbildung 12 veranschaulicht, die auch wie Abbildung 5 aus [Elb17] entnommen wurden.

Ziel dieses Kapitels ist es, die maximale Anzahl von Kandidaten in jeder Kurve zu beschränken unter der Voraussetzung, dass die Güte des resultierenden Circuit-Kandidaten nur um einen Faktor $\varepsilon > 0$ von der optimalen Lösung entfernt ist, mit dem Vorteil, diese Lösung in polynomieller Zeit berechnen zu können.

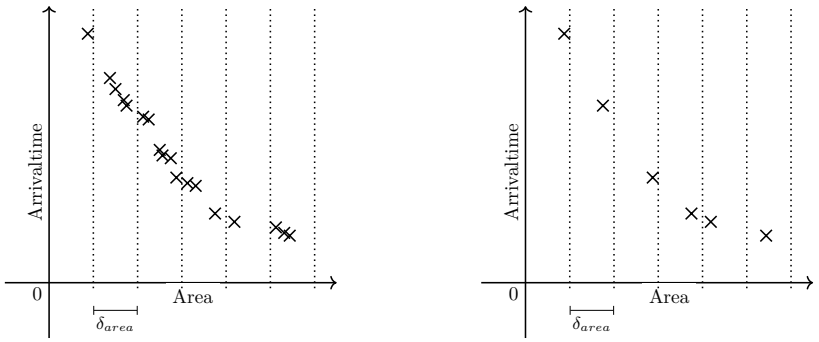


Abbildung 12: Einteilen und Filtern mit Buckets

Man wählt die Bucketgrößen $\delta_{delay} = \frac{\varepsilon X^-}{2n\lambda}, \delta_{area} = \frac{\varepsilon X^-}{2n(1-\lambda)}$ (mit Tradeoff λ und Approximationsfaktor ε) und speichert erst nur den schnellsten Kandidaten in jeder Area-Bucket und von den verbleibenden nur den kleinsten in jeder Delay-Bucket. Hierbei entspricht X^- der mit dem Untere Schranke Area Algorithmus errechneten Schranke und n der Anzahl der Knoten des Circuits.
Für $\lambda \in \{0,1\}$ ist jeweils eine der Bucketgrößen nicht definiert. In diesem Fall wird für den nicht definierten Fall nur der beste Kandidat (schnellster bzw. kleinster) behalten, sodass im nicht definierten Fall die Tradeoffkurve in nur ein Bucket unterteilt wird.
Die maximale Anzahl an Kandidaten in einer Tradeoffkurve ist nach dem Filtern polynomiell beschränkt. Filtert man jede Tradeoffkurve wie beschrieben, ist der noch bestmögliche Circuit Kandidat nur um den Faktor ε von der optimalen Lösung entfernt. Ein Beweis dieser Aussage befindet sich in [Elb17].

Definition 3.11. Sei $\mathcal{B}$ die maximale Anzahl an Kandidaten in einer Tradeoffkurve, nachdem sie gefiltert wurde.
Des Weiteren sei $\mathcal{K}$ die maximale Anzahl von Klassen eines Knotens.

Korollar 3.12. *Für Circuits C mit Library L, konstanter Anzahl k an Multifanoutknoten und* $\max_{g_1,g_2 \in L}\{\max\{\frac{Area(g_1)}{Area(g_2)}, \frac{Delay(g_1)}{Delay(g_2)}\}\}$ *ist beschränkt durch eine konstante c, sind $\mathcal{B}$ und $\mathcal{K}$ polynomiell beschränkt.*

Beweis: $\mathcal{B}$ ist beschränkt durch die Größe des größtmöglichen äquivalenten Circuit C_A von C geteilt durch δ_{area}. Sei X^+ eine mit dem Algorithmus: Untere Schranke Arrivaltime errechnete obere Schranke für die Größe von C. Daraus folgt $\mathcal{B} \in \mathcal{O}(X^+ \cdot \frac{2n(1-\lambda)}{X^-\varepsilon})$. $\mathcal{B}$ ist somit polynomiell in $\frac{X^+n}{X^-\varepsilon}$.
Da die Größe $area_{max}$ (Delay) des größten (langsamsten) Gates der Library weniger als einen Faktor c größer (langsamer) als die entsprechenden Werte des kleinsten (schnellsten) Gates dieser Library ist, folgt, dass $\frac{X^+n}{X^-\varepsilon}$ polynomiell beschränkt ist (Dies ist ebenfalls der Fall für die in Abbildung 2 beschriebene Library). Der größtmögliche zu C äquivalente Circuit ist nämlich nicht größer als $|V(C)|(fanin_{max}+1)area_{max}+|O|area_{max}$. Die Größe jedes Gates mit möglichen Invertern an seinen Inputs in C ist durch $(fanin_{max}+1)area_{max}$ beschränkt und die Anzahl der an den Outputs O liegenden Inverter durch $|O|$.
Für eine allgemeine Library findet sich in [Elb17] eine Lösung zur Erhaltung der Polynomialität.
Sei $v \in V(C)$ und $\mathcal{K}_v$ die Anzahl Klassen von v. Sei des Weiteren $K_{h,i}$ die maximale Anzahl Klassen eines Multifanoutknotens mit i weiteren Multifanoutknoten in seiner Cone. An einem Knoten v ist $\mathcal{K}_v$ beschränkt durch die Anzahl aller möglichen Kombinationen von Kandidaten aller möglicher Klassen der Multifanoutknoten in $cone(v)$. Daraus folgt $\mathcal{K}_v \leq (\mathcal{B}\mathcal{K}_{h,k-1})^k$; da v maximal k Multifanoutknoten in seiner Cone hat, welche jeweils maximal $k-1$ Multifanoutknoten in ihrer Cone haben. Deshalb gilt $\mathcal{K}_{h,k} \leq (\mathcal{B}\mathcal{K}_{h,k-1})^k$.
Es gilt $\mathcal{K}_{h,0} \leq 1$, da der Multifanoutknoten keine offenen Knoten besitzen kann.
Daraus folgt $\mathcal{K}_v \leq \mathcal{B}^{k!}$. Da k beschränkt ist, folgt, dass $\mathcal{K}$ polynomiell beschränkt ist. Hierbei ist sind $\mathcal{B}$ und $\mathcal{K}$ auch in ε polynomiell beschränkt.
□

Beschränkt man die maximale Anzahl von Multifanoutknoten in C, folgt aus Korollar 3.12, dass die Anzahl der Klassen eines jeden Knotens und die Menge der sich darin befindlichen Kandidaten polynomiell beschränkt ist.
Deshalb geht der im Folgenden vorgestellte polynomielle Algorithmus von einer beschränkten Anzahl von Multifanoutknoten aus.

Es folgt ein kurze Beschreibung, wie die Kandidatemenge eines Knotens gebildet wird.

3.6.3 Verknüpfen von Kandidaten

Ohne Multifanoutknoten lassen sich bei der Konstruktion eines Kandidaten für einen Knoten x die Kandidaten k_i der Inputs i unabhängig voneinander wählen. Da das Klonen ausgeschlossen wurde, ist dies bei Circuits mit Knoten höheren Fanouts nicht möglich.
Sei O_v die Menge der offenen Knoten eines Knoten v. Die folgenden Bedingungen stellen sicher, dass Klonen verhindert wird:

1. $\forall v, w \in Inputs(x) \forall y \in O_v \cap O_w : class(k_v)(y) = class(k_w)(y)$
2. $\forall v, w \in inputs(x)$ mit $v \in O_w : k_v = class(k_w)(v)$

Bedingung 1 stellt sicher, dass an den offenen Knoten der Input-Kandidaten ein eindeutiger Kandidat festgelegt wird.
Bedingung 2 sichert diese Eigenschaft auch für die Inputs selber, denn es ist möglich, dass ein Input-Knoten von x auch ein offener Knoten eines weiteren Inputs ist. Für diesen wird dadurch ebenfalls ein eindeutiger Kandidat festgelegt.
Somit sind alle Input-Kandidaten-Mengen, welche diese beiden Bedingungen erfüllen, eine mögliche Grundlage für einen Kandidaten auf x.

3.6.4 Finden von Kandidaten

Für jedes der maximal $\mathcal{M}$ Matche eines Knotens v von einem Circuit mit den oben genannten Einschränkungen gilt Folgendes zur Findung aller passenden Kandidaten:

Jedes Match m besitzt höchstens $fanin_{max}$ Inputs. Sei K_i die Menge der Klassen von Input i des Matches. Jedes Element aus der Menge der möglichen Klassen $\prod_{i \leq |inputs(m)|} K_i$ wird auf die obigen Bedingungen überprüft. Die Kombinationen O, welche beide Bedingungen erfüllen, bilden eine Klasse von v. In die dazugehörige Tradeoffkurve kommt nun jede nicht dominierte Kombination von Kandidaten der Tradeoffkurven von O.
Die lässt sich in Laufzeit $\mathcal{O}(\mathcal{M}(\mathcal{KB})^{fanin_{max}})$ implementieren. Da alle Elemente dieser Formel polynomiell beschränkt sind und $fanin_{max}$ konstant ist, entspricht dies polynomieller Laufzeit.

3.7 FPTAS

Für Circuits mit einem Output, beschränktem $fanin_{max}, fanout_{max}$ ohne teilweise überflüssige Subcircuits und mit maximal k Multifanoutknoten gibt es für das folgende Problem einen FPTAS (fully polynomial time approximation scheme). Ein FPTAS ist ein polynomieller Algorithmus der, gegeben ein beliebiges $\varepsilon > 0$, eine Lösung des Problems in bezüglich ε polynomieller Laufzeit errechnet mit der Eigenschaft, dass für dessen Kosten $c \leq (1+\varepsilon)OPT$ gilt. Hierbei sind OPT die Kosten einer optimalen Lösung des Technology Mappings bezüglich der bisher eingeführten Operationen.
Der diesem Kapitel zugrunde liegende Algorithmus wurde in [Elb17] entwickelt und implementiert.

FPTAS für das Technology Mapping	
Instanz:	Circuit C mit einem Output, Library L mit beschränktem $fanin_{max}$, maximal k Multifanoutknoten mit beschränkten $fanout_{max}$, Tradeoffparameter $\lambda \in [0,1]$, Toleranz $\varepsilon > 0$.
Aufgabe:	Finde einen Circuit-Kandidaten K auf C mit Kosten $\lambda AT(K) + (1-\lambda)Area(K) \leq (1+\varepsilon)OPT$.

Diese Problemstellung lässt sich mit folgendem Algorithmus lösen.

Algorithmus : FPTAS für das TM mit Konvexkombination

Input : Circuit C ohne teilweise überflüssige Subcircuits und mit finalem Output o, Library L mit beschränktem $fanout_{max}$, $\varepsilon > 0$, k Multifanoutknoten mit beschränktem $fanout_{max}$, Tradeoff $\lambda \in [0,1]$

```
1  M ← finde_alle_matche(C)
2  lösche_konst_Kanten_überdeckende_Matche(M, C)
3  X⁻ ← Untere_Schranke_Area(C)
4  foreach Knoten g ∈ V(C) do
5      berechne_offene_Knoten(g)
6  foreach Knoten g ∈ V(C) in topologischer Reihenfolge do
7      g.tradeoff_curves[] ← [∅]
8      foreach Match m ∈ M[g] auf g do
9          foreach mögliche Klasse A auf g do
10             foreach Kandidaten k auf g mit m der A respektiert do
11                 if k ist nicht dominiert in g.tradeoff_curves[A] then
12                     g.tradeoff_curves[A].push_back(k)
13     filter_mit_Buckets(g.tradeoff_curves, X⁻, ε)
14 C' ← Circuit(bester_Kandidat(o, λ))
15 return C'
```

Hierbei wurde der Kern-Algorithmus um die in diesem Kapitel beschriebenen Routinen erweitert.
Für jeden Knoten werden zuerst alle, keine konstanten Kanten überdeckenden, Matche berechnet und seine offenen Knoten ermittelt. Dann werden für alle Knoten in topologischer Reihenfolge und deren Klassen alle nicht dominierten Kandidaten hinzugefügt und die Tradeoffkurven daraufhin gefiltert. Es bleiben für jeden Knoten aufgrund der Beschränkung auf k Multifanoutknoten polynomiell beschränkt viele Kandidaten übrig. So auch für den Output o. Anschließend wird aus o's einziger Tradeoffkurve der beste Kandidat $k*$ ausgewählt und der zu C äquivalente Circuit C' anhand von k^* gebaut und zurückgegeben.

Lemma 3.13. *FPTAS für das TM mit Konvexkombination ist polynomiell in* $\mathcal{O}(|V(C)|^2 + |V(C)|\mathcal{MBK}^2)$ *implementierbar.*

Beweis: Schritt 1 lässt sich nach Kapitel 3.5.2 in polynomieller Zeit implementieren. Schritt 2 ist linear in der polynomiell beschränkten Anzahl an Matches im gesamten Circuit. Die Schritte 4 und 5 lassen sich in $\mathcal{O}(|V(C)|^2)$ errechnen, da von einem Knoten v ausgehend durch Iterieren über die Inputs alle Knoten der Cone hinzugefügt werden. Multifanoutknoten, welche Inputs eines offenen Knotens sind, werden als offen deklariert und Multifanoutknoten, welche einen Output außerhalb der $cone(v)$ besitzen, werden ebenfalls als offen markiert. Da $fanout_{max}$ beschränkt ist, folgt eine lineare Laufzeit in $|V(C)|$ für die Berechnung der offenen Knoten von v. Dies wird für jeden Knoten aus C wiederholt.

Die Schleifendurchläufe der Schritte 6 bis 10 sind beschränkt durch $|V(C)|$, $\mathcal{M}$, $\mathcal{K}$, $\mathcal{B}$. Schritte 11 und 12 sind in $\mathcal{O}(\mathcal{K})$ implementierbar.
Die Kardinalität der Menge der Kandidaten in den Tradeoffkurven ist beschränkt durch die Anzahl der Schleifendurchläufe der Schritte 8-12. Somit liegt die Laufzeit polynomiell in der Anzahl der Schleifendurchläufe.
Schritt 14 ist durch die maximale Anzahl der Kandidaten einer Tradeoffkurve und $|V(C)|$ beschränkt. Schritt 3 liegt in der Summe der Laufzeit der bisher beschriebenen Schritte.
Daraus folgt eine Gesamtlaufzeit von $\mathcal{O}(|V(C)|^2 + |V(C)|\mathcal{MBK}^2)$. □

Aus Korollar 3.12 folgt, dass das FPTAS für das TM mit Konvexkombination auch polynomiell in ε ist und somit ein FPTAS ist.

3.8 Heuristik

Im folgenden Algorithmus, welcher in [Elb17] entwickelt wurde, ist die Menge der Knoten mit $fanout > 1$ beliebig groß. Die Anzahl der Multifanoutknoten bestimmt maßgeblich den Speicherbedarf an Kandidaten und die Laufzeit des FPTAS. Sie sind der Grund, warum sich das FPTAS für eine Anwendung des Technology Mappings auf einem gesamten Chip nicht eignet.
Von daher ist es ein naheliegender Ansatz für eine Heuristik, an jedem Multifanoutknoten nur einen Kandidaten zu speichern. Es hat einen sehr hohen Einfluss auf die Güte der Lösung, welchen man dort auswählt. Routinen für eine solche Auswahl werden in Kapitel 4.2 ausführlich behandelt. Im Folgenden wird beispielhaft eines dieser Verfahren erläutert.
Ist an jedem Multifanoutknoten ein Kandidat gewählt, lässt sich der noch bestmögliche Circuit-Kandidat schnell errechnen, da für jeden Knoten nur eine Klasse mit einer Tradeoffkurve vorhanden ist.

Algorithmus : Heuristik für das TM mit Konvexkombination

Input : Circuit C ohne vollständig überflüssige Subcircuits und mit finalem Output o, Library L, Tradeoff $\lambda \in [0,1]$

```
1  M ← finde_alle_matche(C)
2  lösche_konst_Kanten_überdeckende_Matche(M, C)
3  foreach Knoten g ∈ V(C) do
4      berechne_offene_Knoten(g)
5  foreach Knoten g ∈ V(C) in topologischer Reihenfolge do
6      g.tradeoff_curves[] ← [∅]
7      foreach Match m ∈ M[g] auf g do
8          foreach mögliche Klasse A auf g do
9              foreach Kandidaten k auf g mit m der A respektiert do
10                 if k ist nicht dominiert in g.tradeoff_curves[A] then
11                     g.tradeoff_curves[A].push_back(k)
12     if g ist Multifanoutknoten then
13         guess ← argmin_{Kandidat k auf v} {λAT(k) + (1 − λ)area(k)}
14         foreach Kandidat k auf g do
15             if k ≠ guess then
16                 lösche k
17     filtern_mit_Buckets(g.tradeoff_curves, 0)
18 C' ← circuit(bester_kandidat(o, λ))
19 return entferne_buffer(C')
```

Dieser Algorithmus unterscheidet sich nur in den Schritten 12-17 und 19 von dem FPTAS. Diese Schritte beschreiben ein einfaches Verfahren zur Auswahl eines Kandidaten für einen Multifanoutknoten. Genau wie bei dem Kern-Algorithmus wird nur der Kandidat behalten, welcher den Tradeoff minimiert. Dies garantiert jedoch keine optimale Lösung.
Die Menge an Kandidaten in einer Tradeoffkurve ist in der Praxis durch die Einschränkung auf einen Kandidaten für Multifanoutknoten deutlich geringer als im FPTAS. Aus diesem Grund lässt sich in der Praxis ohne Laufzeiteinbußen mit $\varepsilon = 0$ filtern.
Weitere Informationen und Erweiterungen dieser Routine befinden sich in den Kapiteln 4.2 und 6.
Schritt 19 entfernt die durch teilweise überflüssige Subcircuits evtl. hinzugekommenen Buffer.

Laufzeit: Stellt man die gleichen Voraussetzungen an die Circuits wie der FPTAS und filtert mit einem $\varepsilon > 0$, lässt sich diese Heuristik in derselben polynomiellen Laufzeit implementieren.
In allgemeinen Circuits ist dies wahrscheinlich nicht mehr möglich, denn die Anzahl möglicher Matche steigt durch die teilweise überflüssigen Circuits exponentiell (siehe Kapitel 3.5).

4 Erweiterungen der Heuristik

4.1 DAGs mit mehreren Outputs

Bisher wurde auf Circuits C mit nur einem Output gearbeitet. Instanzen realer Chips sind jedoch mit beliebig vielen Outputs ausgestattet. Outputs können auch zusätzlich noch Nachfolger in C besitzen.

Da Signale von Output-Knoten mit Fanout in C aus der Cone darüberliegender Knoten laufen können, werden sie ebenfalls als offene Knoten ihrer Nachfolger deklariert. In einem Circuit mit mehreren Outputs ist AT alleine ein unzureichendes Optimierungskriterium.

4.1.1 Required Arrivaltimes

In Definition 2.6 wurde bereits der Begriff der Arrivaltime eines Knotens eingeführt. Dies ist die Zeit, zu welcher das letzte Signal bei einem Knoten ankommt. Diese Werte sind für die Input-Knoten eines Circuits C gegeben und werden von dort aus, unter Hinzunahme von Wire-, Gate- und Inverter-Delay, für jeden Knoten von C in topologischer Reihenfolge errechnet.
Im Designprozess eines Chips gibt es neben der tatsächlichen Arrivaltime auch noch eine gewünschte Arrivaltime RAT (required AT), welche an den Outputs eines Graphen gegeben ist und ähnlich zur AT durch C propagiert wird. Somit sind sowohl AT als auch RAT eine Funktion auf $V(C)$.
Der Vollständigkeit wegen folgt die genaue Definition der RAT:

Definition 4.1. RAT:
Sei C ein Circuit und $v \in V(C) \backslash Outputs(C)$. Die RAT (required arrivaltime) an v ist definiert durch:

$$RAT(v) := \min_{\substack{(v,x)\in E(C),\\ i:inputs(x)[i]=v}} \{RAT(x) - d_{w(v,x)} - d_{gate(x)} - d_{inv}\mathbb{1}_{inv_x(i)}\}.$$

Die RAT der Outputs wird hierbei (wie das Delay der Input-Knoten) als gegeben angenommen.

In der Praxis kommen die Signale oft später an als erwartet. Der Betrag des Slack $slack(v) := RAT(v) - AT(v)$ gibt, wenn $slack(v) \leq 0$, an, um wie viel Zeit sich das letzte Signal an v verspätet. Somit ist es viel sinnvoller einen gegebenen Circuit hinsichtlich des negativen Slacks zu verbessern.
Hieraus ergeben sich für einen Circuit die beiden folgenden Werte:

- Worst-Slack (WS): Wert des kleinsten Slacks für einen Knoten auf dem Circuit.
- Sum-of-Negative-Slacks (SNS): Die Summe von allen negativen Slacks der Outputs eines Circuits.

Letzterer hat in der Praxis mehr Bedeutung, da eine sehr gute Verbesserung der SNS eine Verbesserung des WS in der Regel mit einschließt.

Mit dieser Definition lässt sich kurz der Anwendungsbereich des FPTAS in der Logik-Optimierung eines Chips darstellen:
Bei der Analyse eines Chips lässt sich auf diesem ein Knoten v finden, an welchem der WS angenommen wird.
Sei C der Circuit, welcher nur aus dem Gate von v besteht. Füge nun zu v in C den Input von v hinzu, welcher den größten negativen Slack besitzt. Dies wiederhole man für das neu hinzugefügte Gate, bis man an einen Input des Chips gelangt oder das am Input liegende Bauteil nicht in der Library des Technology Mappings liegt.
Hieraus entsteht ein Circuit C, welcher einen Output hat und aus einer hintereinander geschalteten Kette von Knoten besteht. Dieser lässt sich nun mit geeigneten Algorithmen zu einem Pfad, bestehend abwechselnd aus AND und OR Gates, umbauen und zu einem äquivalenten Circuit C' mit geringerer Tiefe (Anzahl Kanten eines bzgl. der Kardinalität der Kantenmenge längsten Weges in C) umformen. Ausführliche Informationen zu den hier zum Einsatz kommenden Algorithmen befinden sich in [WRS07], [BH18] und [HS17].

Diese umgebauten Instanzen besitzen nur einen Output und wenige Multifanoutknoten mit geringem Ausgangsgrad. Dadurch sind diese Instanzen gut für den oben vorgestellten FPTAS geeignet. Der große Vorteil von diesem Vorgehen sind überschaubar große Instanzen und eine Beschleunigung des gesamten Chips durch eine geringe Laufzeit des Algorithmus. Der Nachteil ist jedoch, dass ein Chip oft sehr viele Wege mit einem schlechten negativen Slack hat und man somit den Chip nur inkrementell beschleunigt.

Eine alternative Herangehensweise an das Technology Mapping ist, einen Circuit dahingehend zu optimieren, dass die SNS minimiert wird.
Für $RAT = 0$ ist dies bei den bisher betrachteten Circuits äquivalent zur AT-Optimierung, da nur Instanzen mit einem Output betrachtet wurden.
Die Heuristik wurde im Zuge dieser Arbeit für Instanzen mehrerer Outputs und die Optimierung hinsichtlich der RAT weiterentwickelt und implementiert.

4.1.2 Implementierung mehrerer Outputs

Wie in der Einleitung beschrieben, ist es das Ziel dieser Arbeit eine Heuristik für das Technology Mapping zu entwickeln, welche auf großen Teilen eines Chips lauffähig (bezüglich Laufzeit) ist. Da ein solcher Chip mehr als nur einen Output-Pin hat, lässt er sich in zusammenhängende Circuits unterteilen, welche mehr als einen Output-Knoten haben. Folgende Umbauten sind notwendig, um mit dem Kern-Algorithmus auch diese Instanzen zu verbessern.
Sobald der Algorithmus für jeden Knoten die Kandidatenmenge errechnet hat, lässt sich nicht einfach der beste Kandidat für den Output aus seiner Tradeoff-Kurve auswählen. Dieser besitzt bei mehreren Outputs nämlich in der Regel offene Knoten. Es wurde bereits dargestellt, wie man mehrere Kandidaten auswählt, sodass diese an allen offenen Knoten übereinstimmen. Somit lässt sich ein Circuit mit den Mitteln aus Kapitel 3.6.3 realisieren,

welcher eine Kostenfunktion hinsichtlich Größe und WS optimiert.

Wie bereits in Kapitel 4.1.1 erläutert, ist es in der Praxis profitabler, die SNS anstatt den WS des Circuits zu verbessern.
Dementsprechend muss aus den Kandidatenmengen der Outputs derjenige Circuit-Kandidat gebaut werden, welcher die SNS minimiert.
Dieses Kriterium ersetzt von diesem Punkt an das Kriterium der Delay-Optimierung in der Kostenfunktion.

Des Weiteren müssen nach dem Technology Mapping noch alle Outputs mit der bis zu ihnen berechneten logischen Funktion vorhanden sein. Daraus folgt, dass über einen Output-Knoten, welcher in dem Circuit noch mindestens einen Nachfolger hat, nicht gematcht werden darf, weil sonst ein nicht erlaubter Seitenoutput entstehen würde.
Dies lässt sich dadurch sicherstellen, dass man eine seiner ausgehenden Kanten als konstant deklariert, wie das bereits bei den zu langen Kanten geschehen ist.

4.2 Premapping von Multifanoutknoten

Der exponentielle Anstieg der Kandidatenmenge wird, wie in Kapitel 3.6 gezeigt, durch die Multifanoutknoten verursacht, wodurch ein sehr hohes Laufzeit-Potenzial in der Reduzierung der Kandidaten für diese Knoten liegt. Dieses Kapitel stellt mehrere Routinen einer solchen Reduzierung vor.

Die Kandidatenmenge eines jeden Multifanoutknotens wird, wie bereits in Kapitel 3.8 dargestellt, auf eins reduziert. Diese Routine wird auch das Premapping der Multifanoutknoten genannt. Hieraus folgt, dass jeder Knoten des Circuits C nur noch eine Klasse an Kandidaten besitzt, denn alle offenen Knoten seiner Cone sind Multifanoutknoten und somit festgelegt. Die Kandidatenmenge der Outputs, welche noch Nachfolger $o \in C$ haben, wird ebenfalls auf einen Kandidaten reduziert. Dies verhindert das Klonen in $cone(o)$, da auch die nicht offenen Knoten von o von allen Knoten der Menge $O := \{v \in Outputs(C) : o \in cone(v)\}$ mit einem Kandidaten belegt werden.
Nun lässt sich der bestmögliche Kandidat eines jeden Outputs ohne Nachfolger finden, indem in der einzig verbleibenden Tradeoffkurve nach dem Kandidaten mit den geringsten Kosten gesucht wird. Dadurch ist das Finden des bestmöglichen Circuit Kandidaten, welcher keine der gelöschten Kandidaten benutzt, ohne Laufzeiteinbußen möglich.
Daraus folgt, dass der zu wählende Circuit-Kandidat eindeutig ist, sobald jedem Multifanoutknoten ein Kandidat zugewiesen wurde. Aus diesem Grund hat die Wahl der Premapping Routine eine zentrale Bedeutung in der Heuristik.

Im Folgenden werden drei verschiedene Routinen vorgestellt und auf deren Eigenschaften eingegangen. Weitergehende Analysen bezüglich der Unterschiede zwischen den Resultaten von zwei dieser Routinen sind in Kapitel 6 dargestellt.

4.2.1 Triviales Premapping

Diese Methode des Premappings wurde bereits in Kapitel 3.8 angewandt. Hierbei wird für jeden Knoten folgender Kandidat ausgewählt:

$$guess \leftarrow \underset{\text{Kandidat } k \text{ auf } v}{\arg\min} \{\lambda AT(k) + (1-\lambda) area(k)\}.$$

Diese Methode lässt sich jedoch wie nachfolgend beschrieben verbessern.

Im Gegensatz zu dem Kern-Algorithmus aus Kapitel 2.2 kann auch mit $\lambda \in \{0,1\}$ keine optimale Lösung garantiert werden. Jedoch befindet sich die dadurch gefundene Lösung nahe an der optimalen Lösung. Die Analyse, wie weit eine solche Lösung von der optimalen Lösung entfernt ist, wird in Kapitel 6.2.4 dargestellt.

4.2.2 Premapping durch Schätzen

Beim Premapping durch Schätzen wird eine Vermutung für die Kosten des Kandidaten eines jeden Multifanoutknoten in einer optimalen Lösung des Technology Mappings aufgestellt. Ausgewählt wird dann der Kandidat, welcher die geringste Differenz zu den vermuteten Kosten besitzt.
Die im Folgenden vorgestellte Routine wurde durch [Elb17] für einen Output entwickelt und wird hier auf SNS erweitert.
Die Schätzung erfolgt durch zwei Technology Mapping Läufe, welche einmal einen möglichst schnellen und einmal einen möglichst kleinen äquivalenten Circuit errechnen.

Folgender Algorithmus, der in [Elb17] entwickelt wurde, nähert die entsprechenden Circuit-Kandidaten an:

Algorithmus : Heuristische untere Schranke Arrivaltime

```
   Input : Circuit C, Library L
 1 schnellster_kandidat[] ← ∅
 2 schnellster_inv_kandidat[]← ∅
 3 foreach Knoten v ∈ V(G) in topologischer Reihenfolge do
 4 |   berechne_alle_(invertierten)_Matche(v)
 5 |   foreach Match m auf v do
 6 |   |   k, k_inv ←berechne_schnellsten_(inv)_Kandidaten(m,v)
 7 |   |   Update_schnellster_(inv)_kandidat(k, k_inv)
 8 |   if v ist Multifanoutknoten then
 9 |   |   if schnellster_kandidat[v].AT <
   |   |   schnellster_inv_kandidat[v].AT then
10 |   |   |   lösche schnellster_inv_kandidat[v]
11 |   |   else
12 |   |   |   lösche schnellster_kandidat[v]
13 Baue C' entsprechend schnellster_kandidat[o]
14 return C'
```

Dieser Algorithmus führt den Kern-Algorithmus auf C aus und behält für jeden Knoten nur einen Kandidaten. Dadurch ist es nicht notwendig, mehr als $|V(C)|$ Kandidaten zu speichern.
Die untere Schranke für Area wird mit einem entsprechenden Algorithmus berechnet, indem an jedem Knoten der kleinste Kandidat, anstelle des schnellsten, gespeichert wird. Der entsprechende Algorithmus wird Heuristische untere Schranke Area genannt.
Im Gegensatz zu dem Kern-Algorithmus garantiert dieser keine optimale Lösung bezüglich Arrivaltime oder Area, da es möglich ist, dass für einen Knoten v und einen seiner Inputs i der schnellste bzw. kleinste Kandidat auf v eine Invertierung an i vorsieht. Da der beste Kandidat mit invertiertem Output von i evtl. in Schritt 10 oder 12 gelöscht wurde, ist der optimale Kandidat für v möglicherweise nicht mehr vorhanden. Der Unterschied zum Kern-Algorithmus liegt darin, dass dort für i die besten Kandidaten beider Invertierungen gespeichert werden. Da i hier jedoch evtl. ein Multifanoutknoten ist, würde dies zu einem Verlust der Polynomialität führen.
Im Gegensatz zu dem Algorithmus aus Kapitel 3.6.1, mit welchen die Errechnung einer unteren und oberen Schranke für die AT und $Area$ möglich ist, gibt dieser Algorithmus einen äquivalenten Circuit zurück.
Die Laufzeit dieses Algorithmus liegt in der Laufzeit der Heuristik aus Kapitel 3.8. Da bei der zweimaligen Ausführung dieser Routine nur maximal zwei Kandidaten an jedem Knoten gespeichert werden und alle Matche nur einmal gefunden werden müssen, wird die Laufzeit der Heuristik mit der Hinzunahme dieses Algorithmus nur geringfügig beeinflusst.

Das Premapping durch Schätzen erfolgt nun durch folgenden Algorithmus:

Algorithmus : Premapping durch Schätzen

```
1  C'_AT ←Heuristische_untere_Schranke_Arrivaltime(C)
2  C'_Area ← Heuristische_untere_Schranke_Area(C)
3  foreach Output o do
4      α[o] ← λ · AT_{C'_AT}(o) + (1 − λ) · AT_{C'_Area}(o) //Finale AT Schätzung
5  foreach Multifanoutknoten v ∈ V(C) do
6      estim_small(v) ←neg_Slack(v, C'_Area, α)
7      estim_fast(v) ←neg_Slack(v, C'_SNS, α)
8      neg_Slack_Schätzung(v) ← λ·estim_fast(v) + (1 − λ)·estim_small(v)
9
10 ...
11
12 if v ist Multifanoutknoten then
13     min ← min_{Kandidat k auf v} {|neg_Slack(k) − neg_Slack_Schätzung(v)|}
14     foreach Kandidat k auf v do
15         if |neg_Slack(k)-neg_Slack_Schätzung(v)| ≠ min then
16             lösche k
17     if neg_Slack_Schätzung(v) = 0 then
18         lösche_alle_Kandidaten_bis_auf_den_Kleinsten(v)
```

Die Schritte 1-9 werden an den Anfang der Heuristik für das TM mit Konvexkombination gesetzt. Die Schritte 13-18 ersetzen die Schritte 12-16 der Heuristik.
Es ist ausreichend, alle möglichen Matche der Knoten nur einmal zu errechnen und diese Werte sowohl für die untere Schranke als auch für die Heuristik zu nutzen. Mit diesem Trick lassen sich beide Schranken in kurzer Zeit errechnen. Genauere Informationen zur Laufzeit und Güte der Premapping Routinen befinden sich in Kapitel 6.
Diese Routine liefert in der Praxis sehr gute Ergebnisse, da durch die Berechnung der Schranken das Potenzial eines Knotens errechnet wird (wie klein oder schnell Kandidaten dieses Knotens werden können). Mithilfe des Tradeoffparameters errechnet sich hieraus ein geschätzter Kostenwert. Es wird nur der Kandidat ausgewählt, welcher die Differenz zu dieser Schätzung minimiert. Falls die Slack-Schätzung Null ist, wird nur der kleinste Kandidat mit Slack Null gespeichert, da dieser die anderen aus der Sicht des Slacks dominiert.

4.2.3 Erweitertes Premapping durch Schätzen

Im Zuge dieser Arbeit wurde noch folgende Premapping Methode entwickelt. Es handelt sich dabei um eine Erweiterung der gerade beschriebenen Methode.
Das Premappping durch Schätzen errechnet einen kleinen und einen schnellen Circuit und daraus das Potenzial eines jeden Knotens bezüglich der Verbesserung hinsichtlich Größe und Geschwindigkeit. Der folgende Ansatz findet besonders Delay kritische Pfade in dem Circuit und ordnet dessen

Knoten eine Kritikalität zu. Diese gibt Auskunft über den Einfluss eines Knotens bezüglich des Delays seiner Nachfolger. Das erweiterte Premapping durch Schätzen verknüpft das Potenzial und die Kritikalität eines jeden Knotens bei der Auswahl eines geeigneten Kandidaten.

Jeder Knoten g in C erhält eine Kritikalität $krit_g$. Diese wird für jeden Knoten mit Null initialisiert.
Die Kritikalität wird mithilfe des Dijkstras Algorithmus errechnet. Dieser ist, da ein Circuit C ein kreisfreier gerichteter Graph ist, in der Lage, längste Wege bezüglich der Kantengewichte in C zu finden. Hierbei ist das Gewicht einer Kante das Delay dieser inklusive des Gate-Delays des am Ende dieser Kante liegenden Gates (zuzüglich des evtl. am Input liegenden Inverters). Das Gewicht der von einem Input von C ausgehenden Kanten wird noch um die Arrivaltime des entsprechenden Inputs erhöht.
Der Dijkstra Algorithmus wird für jeden Output o aufgerufen und $krit_g$ wird für jeden Knoten, welcher auf einem längsten Weg zu o liegt, um eins erhöht.

Für jeden Knoten ist nun gespeichert, für wie viele Outputs dieser auf einem kritischen Pfad liegt. Jeder Kritikalitätswert wird nun durch die Anzahl der Outputs dividiert, wodurch diese Werte in $[0,1]$ liegen.

Ziel ist es, nun auf Grund der Information der Kritikalität an jedem Multifanoutknoten v einen Faktor μ_v zu berechnen, mit welchen im Technology Mapping Algorithmus der folgende Kandidat k für v gewählt wird:

$$\min_{\text{Kandidat } k \text{ auf } v} \{|neg\_slack(k) - (1+\mu_v)\text{neg\_slack\_schätzung}(v)|\}.$$

Für den Fall, dass die Slack-Schätzung Null ist, wird auch hier der kleinste Kandidat mit Slack Null gewählt.

Der restliche Teil dieses Kapitel beschreibt eine Funktion für die Berechnung von μ_v.
Im Folgenden sei $f : [0,1] \to [-\mu_{max}, \mu_{max}]$ die μ_v für jeden Knoten v gegeben $krit_v$ berechnende Funktion. Hierbei sei μ_{max} eine obere Schranke für den Betrag von μ_v. Diese wird heuristisch festgelegt.
Um die Auswahl des Kandidaten sinnvoll zu beeinflussen, gilt für f und zwei Kritikalitäten $krit_1$ und $krit_2$:

$$f(krit_1) > f(krit_2) \Leftrightarrow krit_1 < krit_2.$$

Sonst würde für einen bereits Delay-kritischen Knoten ein langsamerer Kandidat gewählt als das Premapping durch Schätzen vorschlägt. Dies führt mit hoher Wahrscheinlichkeit zu einem äquivalenten Circuit mit schlechterer Güte.
Sei $krit_{av} \in \mathbb{R}$ das arithmetische Mittel von allen errechneten Kritikalitäten.

Sei f wie folgt definiert:

$$f(x) := \begin{cases} \frac{\mu_{max}}{krit_{av}-1} \cdot x + \mu_{max}(\frac{1}{1-krit_{av}} - 1) \text{ wenn } x \geq krit_{av} \\ \frac{-\mu_{max}}{1-krit_{av}} \cdot x + \mu_{max}(\frac{1}{1+krit_{av}} - 1) \text{ wenn } x < krit_{av} \end{cases}.$$

Somit ist f aus zwei linearen Funktionen aufgebaut und lässt sich dadurch in konstanter Laufzeit errechnen. Abbildung 13 visualisiert f.

Das erweiterte Premapping durch Schätzen ist in der Lage, die Lösungen, welche mit dem Premapping durch Schätzen errechnet wurden, zu verbessern, jedoch geschieht dies nicht regelmäßig. Des Weiteren verursacht der Dijkstra Algorithmus einen nicht zu vernachlässigen Anstieg der Laufzeit.
Somit bleibt es offen, f, μ_{max} und $krit_{av}$ so zu wählen, dass eine regelmäßige Verbesserung eintritt. Des Weiteren würde ein Dijkstra Aufruf für jeden Knoten oder eine Beachtung von allen nahezu längsten Wegen eine größere Fülle an Informationen liefern. Da es jedoch nur exponentiell beschränkt viele Wege in C gibt, führt dies zu deutlichen Laufzeiteinbußen.

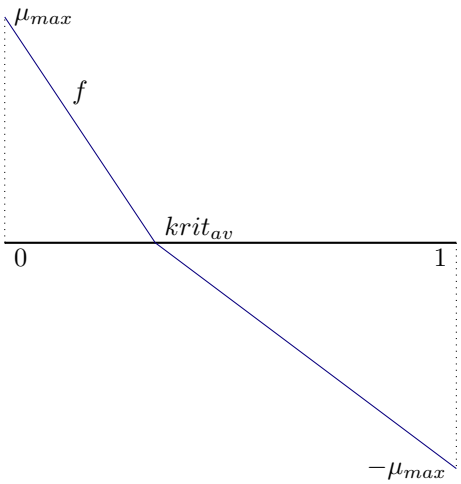


Abbildung 13: Die μ berechnende Funktion f

Diese Premapping Methode wird in Kapitel 6 aufgrund der nur unregelmäßig auftretenden Verbesserungen nicht behandelt.

4.3 Preprocessing

Die Möglichkeiten durch das Matchen sind prinzipiell vielfältig, jedoch bei Gates p mit $|inputs(p)| = fanin_{max}$ auf das beliebige Invertieren der Inputs und des Outputs beschränkt.
Um diesem Problem aus dem Weg zu gehen, ist es sinnvoll, vor dem Technology Mapping Algorithmus jedes Gate mit mehr als zwei eingehenden Kanten durch einen kleinen Subcircuit, bestehend aus zwei Input Gates, zu ersetzen. Dies ist immer möglich, da jede logische Funktion nur mithilfe von NAND2 und INV Gates realisierbar ist ([Pos41]) und auf jedem realen Chip standardmäßig ein AND oder NAND sowie ein OR oder NOR in der Library vorhanden sind. INV Gates sind fester Bestandteil jeder realen Library.
Dabei werden Gates mit hohem Eingangsgrad, wie in Kapitel 2 von [Wer07] beschrieben, zerlegt.
Der Vorteil des Preprocessings ist, dass die Möglichkeiten des Technology Mappings deutlich erweitert werden. Jedoch werden für ein AND4 Gate beispielsweise 3 AND2 Gates eingesetzt, was dazu führt, dass sich meist die Kosten des Ausgangscircuits verschlechtern.
Eine ausführliche Analyse der Vor- und Nachteile des Decomposens findet sich in Kapitel 6.

4.4 Weitere Optimierungskriterien

Das Technology Mapping arbeitet im Chip-Design auf Instanzen realer Chips. Dadurch kommen zu den bereits vorgestellten weitere mögliche Optimierungskriterien hinzu. Es handelt sich hierbei um Kriterien, welche in die Kostenfunktion mit eingebracht werden können und somit während des Technology Mappings optimiert werden. Des Weiteren verursacht der neu implementierte Circuit Kosten, welche im Technology Mapping nicht beachtet wurden, für die es jedoch einen übermäßigen Anstieg zu vermeiden gilt. Ein Beispiel hierfür sind die bereits erwähnten zu langen Kanten.
Darüber hinaus sind viele Gates teilweise symmetrisch aufgebaut (Bsp.: AND, OR ...) und die Signale der Inputs brauchen unterschiedlich lange zum Output des Gates. Daraus folgt, dass durch Permutierung von Teilmengen der Inputs, Geschwindigkeitsvorteile geschaffen werden können.

4.4.1 Pinabhängiges Delay

Bis hierhin war das Delay eines Gates als eine nicht negative reelle Zahl definiert. Die meisten Gates besitzen jedoch mehr als nur einen Input. Die Signale der Inputs brauchen nicht alle dieselbe Zeit um zum Output zu gelangen. Logisch werden die Signale der Inputs zwar alle miteinander verrechnet, jedoch geschieht dies physikalisch nicht gleichzeitig und somit müssen nicht alle Signale zur selben Zeit an den Inputs anliegen.
Die Möglichkeit eines Signals, später an einem Input des Gates ankommen zu können, lässt sich durch einen kleineren Delaywert, spezifisch für diesen Input, realisieren. Denn, wenn das Signal schneller durch das Gate gelangen kann, braucht es auch nicht so früh vorhanden zu sein wie die anderen.
Von nun an ist das Delay eines Gates g: $d_g \in \mathbb{R}_{\geq 0}^{arity(g)}$. Für das Technology Mapping ist dies eine weitere Möglichkeit der Verbesserung, denn viele Gates der Library besitzen mindestens eine Teilmenge von Inputs, welche logisch symmetrisch aufgebaut sind. Diese lassen sich beliebig permutieren. Durch die unterschiedlichen Delay-Eigenschaften der Inputs kann eine solche Permutierung das Delay des Outputs verbessern. Aus diesem Grund ändert sich die AT eines Knotens wie folgt:

Definition 4.2. AT mit pinabhängigem Delay:
Sei C ein Circuit und $v \in V(C)$.
Die AT von v mit pinabhängigem Delay ist wie folgt definiert:

$$AT_p(v) := \max_{i \in inputs(v)} \{d_{gate(v),i} + \mathbb{1}_{\{inv_g(i)\}} d_i + AT_p(i) + d_{w(k,i)}\}\}.$$

Im Folgenden sei mit AT immer das pinabhängige Delay gemeint.
In einem Match ist diese Information bereits abgespeichert, da die Inputs eines Matches mit einer Bijektion an Knoten des Circuits geknüpft werden. Es wird ein Kandidat für jede mögliche Permutation der Inputs gespeichert, falls dieser nicht dominiert ist.
Nach aktuellem Stand gilt $fanin_{max} \leq 4$. Das ist klein genug, um auch bei der Heuristik die max $fanin_{max}$ Permutation bei der Wahl eines Matches in Betracht zu ziehen.

4.4.2 Power-Optimierung

Jedes Gate besitzt neben seinen spezifischen Eigenschaften bezüglich Area und AT noch weitere physikalische Eigenschaften.
An einem Transistor liegt immer eine Spannung an. Daraus folgt, dass dieser auch ohne zu schalten Energie (Power) verbraucht. Diese lässt sich einteilen in static Power und dynamic Power. Hierbei bezeichnet static Power den Energieverbrauch unabhängig von der Benutzung des Transistors. Ein Transistor verbraucht jedoch mehr Energie, wenn er schaltet. Daraus folgt, dass der Energiebedarf abhängig vom Grad der Nutzung dieses Transistors ist. Dieser variable Energieverbrauch wird auch als dynamic Power bezeichnet.
Da ein Gate aus einer logischen Verknüpfung von Transistoren besteht, hat es ebenfalls einen static Power Wert. Da der dynamic Power Wert eines Gates abhängig von der aktuellen Implementierung des Chip ist, ist es schwer, diesen unabhängig von einem Circuit zu errechnen. Aus diesem Grund beschränke ich mich im Folgenden auf die static Power.
Die static Power eines Gates korreliert sehr stark mit der physikalischen Größe dieses Bauteils. Aus diesem Grund lassen sich in den oben vorgestellten Algorithmen die Area Daten durch die static Power Werte ersetzen. Der Circuit wird dadurch hinsichtlich Geschwindigkeit und Energiebedarf optimiert.
Static Power lässt sich auch zusätzlich zu Area in die Kostenfunktion einbauen. Dies würde bedeuten, dass die Kosten eines Kandidaten k das Tripel $area(k)$, $AT(k)$, $static\_Power(k)$ sind, wobei $static\_Power(k)$ ähnlich wie $area(k)$ errechnet wird. Dies führt jedoch zu einer noch schlechteren Vergleichbarkeit von Kandidaten gleicher und verschiedener Knoten. Aus diesem Grund wird auf die Optimierung aller drei Kriterien zusammen verzichtet.
Weitere ausführliche Informationen über Power-Optimierung befinden sich in [Dab17].
In Kapitel 6 finden sich weitere Ausführungen über die Auswirkungen von diesem Austausch in der Kostenfunktion.

5 Resource Sharing

In diesem Kapitel wird ein weiterer Ansatz für eine Heuristik vorgestellt.
Es folgt die allgemeine Definition des Resource Sharing Problems und darauf basierend eine Heuristik, welche sich den Resource Sharing Algorithmus zunutze macht.

5.1 Definition und Lösungsansatz des Resource Sharing Problems

Eine Instanz des Resource Sharing Problems besteht aus einer endlichen Menge von Kunden $\mathcal{C}$ und einer konvexen Menge $\mathcal{B}_c$ für jeden Kunden $c \in \mathcal{C}$. Diese wird auch Block genannt. Des Weiteren sei $\mathcal{R}$ eine endliche Menge von Ressourcen. Gegeben seien des Weiteren Funktionen $g_c : \mathcal{B}_c \to \mathbb{R}^{\mathcal{R}_+}$, welche für jeden Kunden und jedes Element des entsprechenden Blocks einen Ressourcenverbrauch angeben.

Ziel des Resource Sharing ist es, jedem Kunden $c \in \mathcal{C}$ ein Element $b_c \in \mathcal{B}_c$ zuzuordnen, sodass

$$\lambda^* := \inf\{\max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} (g_c(b_c))_r | b_c \in \mathcal{B}_c \forall c \in \mathcal{C}\}$$

approximativ erreicht wird. Mit anderen Worten ist es das Ziel, den maximalen Ressourcenverbrauch zu minimieren.

5.1.1 Lösungsansatz

Der in [MRV11] ausführlich beschriebene Lösungsansatz für das Resource Sharing Problem iteriert über die Kundenmenge und errechnet in jeder Iteration mit einem Orakel und Preisen ω_r für alle Ressourcen $r \in \mathcal{R}$ ein Block Element für jeden Kunden.
Das Orakel $f_c : \mathbb{R}_+^{\mathcal{R}} \to \mathcal{B}_c$, auch Blocklöser genannt, wird als gegeben betrachtet und errechnet für gegebene Preise ω für jedes Element $c \in C$ ein $b_c \in \mathcal{B}_c$ sodass gilt: $\omega^T g_c(b_c) \leq \sigma \inf\{\omega^T g_c(b) | b \in \mathcal{B}_c\}$ mit einem Approximationsfaktor $\sigma \geq 1$.
Nachdem der Blocklöser aufgerufen wurde, wird geprüft, ob verknüpfte Ergebnisse der bisherigen Iterationen eine zulässige Lösung der Resource Sharing Instanz liefert. Ist dies nicht der Fall, so werden die Preise angepasst und die nächste Iteration mit den neuen Preisen gestartet. Sei y_r die Summe des Ressourcenverbrauchs der Ressource r der vorangegangenen Iterationen. Die Preise werden kontinuierlich durch $\omega_r \to \omega_r \cdot e^{\gamma \cdot y_r}$ aktualisiert. Hierbei ist $\gamma > 0$ beliebig, hat jedoch Einfluss auf die Güte der Lösung des Algorithmus.

Ist der Algorithmus fertig, liegt das Ergebnis für jeden Kunden in Form einer Konvexkombination von Elementen des zugehörigen Blocks vor. Da jeder Block konvex ist, repräsentiert diese Kombination wieder ein Element des Blocks.
Angenommen ein Block ist nicht konvex, muss eine Lösung des Resource Sharing noch zu einem geeigneten Element des zugehörigen Blocks gerundet werden.

Das Resource Sharing Problem lässt sich mit einem FPTAS lösen. Ein Beweis hierzu befindet sich in [MRV11].

5.2 Resource Sharing und Technology Mapping

Der folgende Algorithmus wurde in dieser Arbeit angelehnt an [DHHS18] entwickelt.
Das Technology Mapping Problem wird wie folgt durch eine Instanz des Resource Sharing modelliert.
Die Technology Mapping Instanz bestehe aus einem Circuit G ohne teilweise überflüssige Subcircuits und gegebenen RATs an den Outputs von G.
Gesucht sei ein zu G äquivalenter Graph $G*$, dessen Outputsignale vor der entsprechenden RAT ankommen und $Power(G*) \leq B$. Sei B vorerst gegeben.
Sei T_G der Timing Graph von G. Dieser modelliert alle Delays der Knoten und Kanten von G. Dieser entsteht aus G, indem alle Gate-Knoten $g \in V(G)$

durch $fanin(gate(g))+1$ Knoten ersetzt werden. Jede der Input-Kanten von g wird durch einen Knoten unterteilt. Alle Delay-Werte von G lassen sich nun auf T modellieren, indem für jede Kante aus G der entsprechenden Kante in T das Delay dieser Kante zugewiesen wird. Für jeden Gate-Knoten g aus G wird den entsprechenden Kanten das pinabhängige Delay zugeordnet. Ein Beispiel eines solchen Graphen befindet sich in Abbildung 14.
Sei nun $\mathcal{C} := \{G\}$, $\mathcal{B}_G$ die Menge der nicht dominierten Circuit Kandidaten auf G und $\mathcal{P}_G$ die Menge der inklusionsmaximalen Wege im Timing Graphen von G. Die Signale der Outputs eines Graphen kommen genau dann vor der RAT an, wenn alle $P \in \mathcal{P}$ vor der RAT des zu P gehörenden Outputs ankommen.
Somit lässt sich wie folgt festlegen: $\mathcal{R} = \{Power\} \cup \mathcal{P}_\mathcal{G}$.
Das Orakel in jeder Iteration wird durch eine noch zu beschreibende Technology Mapping Heuristik realisiert.
Das Ziel dieses Orakels ist, den gewichteten Ressourcenverbrauch

$$\omega_{Power} \frac{Power(G)}{B} + \sum_{P \in \mathcal{P}} \omega_P \frac{\sum_{e \in E(P)} Delay_e(G)}{RAT_P} \tag{1}$$

zu minimieren. Hierbei ist RAT_P die RAT des Outputs von P.
Durch diese Modellierung des Resource Sharing ergeben sich die folgenden Probleme:

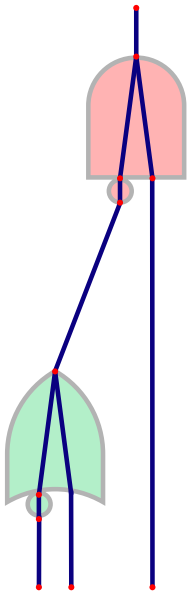


Abbildung 14: Beispiel eines Timing Graphen

- Der Blocklösende Algorithmus (die Technology Mapping Heuristik), liefert einen zu G äquivalenten Circuit G'. Im Allgemeinen gilt jedoch $\mathcal{P}_G \neq \mathcal{P}_{G'}$. Somit lassen sich keine Ressourcenverbräuche von G' auf den Ressourcen vor dem Technology Mapping angeben.
- Der Block $\mathcal{B}_G$ ist nicht konvex, wodurch noch eine geeignete Routine zur Rundung anzugeben ist.

Diese Probleme werden in den folgenden Unterkapiteln gelöst.

5.2.1 Kantenpreise

Der im Folgenden vorgestellte Algorithmus wurde bereits in [DHHS18] entwickelt.
Die Menge $\mathcal{P}_G$ für einen Circuit G ist nur exponentiell beschränkt. Dadurch ist es nicht praktikabel, Preise für jede Pfad Ressource zu speichern. Es ist jedoch möglich, die Preise aller Pfade durch Gewichte auf den Kanten von T_G zu repräsentieren.
Seien $(\omega_P)_{P \in \mathcal{P}_G}$ Preise für die Pfad-Ressourcen von G. Setze $\omega_e := \sum_{P \in \mathcal{P}: e \in P} \omega_P$.

Die Minimierung von (1) durch das Orakel ist nun äquivalent der Minimierung von

$$\omega_{Power}\frac{Power(G)}{B}+\sum_{e\in E(T_G)}\omega_e\frac{Delay_e(G)}{RAT_P}. \tag{2}$$

Des Weiteren gilt für $(\omega_e)_{e\in T_G}$ die Flusserhaltung $\sum_{e\in\delta^-(v)}\omega_e=\sum_{e\in\delta^+(v)}\omega_e$.

Der Preis eines Pfades lässt sich nun in linearer Zeit errechnen. Ein Beweis dieser Aussage befindet sich in [Häh15].
Neben den Preisen der Pfade lässt sich auch die Summe der Ressourcenverbräuche y_p eines Pfades in Verbräuche der Kanten des Pfades aufteilen. Hierbei gilt:

$$y_e := \frac{1}{RAT_P}\sum_{k=1}^{i}\xi^{(k)}\cdot usg^{(k)}(e). \tag{3}$$

Hierbei sind $\xi^{(k)}$ in Iteration k des Algorithmus errechnete Werte in $[0,1]$ und $usg^{(k)}(e)$ entspricht dem der Kante e durch den Algorithmus im folgenden Unterkapitel zugewiesenen Delay.
Auf Basis der Verbräuche auf jeder Kante, lassen sich im Resource Sharing Algorithmus die Pfadpreise, repräsentiert durch Kantenpreise, aktualisieren.

Der, in [DHHS18] entwickelte, Algorithmus Edge Weights aktualisiert die Kantenpreise bei gegebenen Ressourcenverbräuchen $(y_e)_{e\in E(T_G)}$ und γ in linearer Laufzeit. Hierbei gilt für alle Kantenpreise: $\omega_e > 0$.

5.2.2 Technology Mapping Orakel

Das Orakel ist ein Technology Mapping Algorithmus mit den folgenden Besonderheiten:

Zugrunde gelegt wird die Technology Mapping Heuristik aus Kapitel 3.8 mit dem Premapping durch Schätzen und erweitert für Circuits mit mehreren Outputs. Die Kostenfunktionen nutzt hierbei einen Tradeoff von 0,5, denn durch Kantenpreise wird im Laufe der Iterationen des Resource Sharing automatisch auf die Einhaltung der Schranken optimiert.
Damit dieser Algorithmus eine bezüglich der Kantenpreise optimierte Lösung berechnen kann, werden diese wie folgt in den Algorithmus integriert:
Das Delay einer Kante von G wird multipliziert mit dem entsprechenden Preis dieser Kante. Dies gilt ebenfalls für das pinabhängige Delay eines Gates. Das Delay jedes Inputpins eines Gates in G zu seinem Output lässt sich ebenfalls eindeutig einer Kante aus T_G zuordnen. Mit dessen Preis wird auch dieses Delay multipliziert. Des Weiteren werden die Powerwerte der Gates mit ω_{Power} multipliziert. Da $\omega_e > 0$ für jede Kante gilt, führt dies zu positiven Delay-Werten. Durch diese Änderungen minimiert das Orakel heuristisch die Kostenfunktion (2).

5.2.3 Ressourcenproblem

Das Technology Mapping Orakel gibt einen optimierten zu G äquivalenten Circuit G' zurück. Dieser besitzt jedoch neue Pfade und somit eine andere

Ressourcenmenge als G. Pfade aus G' lassen sich jedoch denen aus G zuordnen. Dadurch können die gegebenen Ressourcenverbräuche $(y_e)_{e \in E(T_G)}$ nach folgender, im Zuge dieser Arbeit entwickelter, Routine aktualisiert werden:

Lemma 5.1. *Die Ressourcenverbräuche $(y_e)_{e \in E(T_G)}$ lassen sich in polynomieller Laufzeit anhand des zu G äquivalenten Circuits G' aktualisieren.*

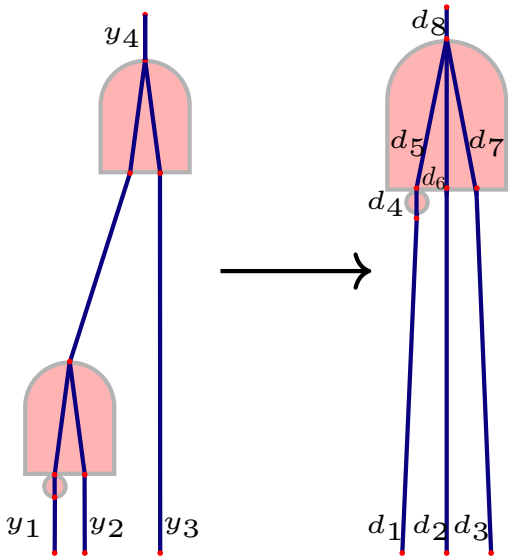
Beweis: G' lässt sich, durch den Circuit-Kandidaten, in eine Menge von Matchen aufteilen. Jeder Pfad von G muss nun durch die auf ihm gebildeten Matche aktualisiert werden. Für jedes Match gilt:
Jeder Pfad durch ein Match, läuft durch genau eine der eingehenden Kanten des Matches und der Weg eines Pfades durch das Match ist eindeutig, sobald diese Kante bekannt ist. Dadurch lässt sich der Ressourcenverbrauch des gesamten Weges durch das Match für alle Pfade dieser eingehenden Kante auf eben dieser eingehenden modellieren. Die eingehenden Kanten eines Matches in G' lassen sich eindeutig Kanten in G zuordnen.
Seien $d_1 ... d_p$ das Delay der Kanten eines solchen Pfades durch ein Match im Timing Graphen von G'. Dieser Pfad ist mit einer eingehenden Kante e des Matches verbunden. Die Kante e existiert bereits in G. Der Ressourcenverbrauch y_e wird nun wie folgt zu y'_e aktualisiert:

$$y'_e := y_e + \frac{\xi \cdot (\sum_{j=1}^{p} d_j)}{RAT}$$

Jeder der angegebenen Ressourcenverbräuche y wird also anhand der Summe der Delay-Werte des zu y korrespondierenden Weges durch das Match aktualisiert. Für den Fall, dass mehrere ausgehende Kanten eines Knotens zu einer zusammengefasst wurden (Bsp.: Mux-Match), werden beide durch die gleiche Summe der Delays aktualisiert. Durch diese Vorgehensweise werden die Ressourcenverbräuche von jedem Pfad im Timing Graphen von G mit der Summe der Delays der Matche in G' auf diesem Pfad aktualisiert. Diese neuen Verbräuche sind auch nach dieser Aktualisierung durch Kanten-Verbräuche in Form von (3) vorhanden.
Diese Aktualisierung lässt sich in polynomieller Laufzeit realisieren, da die Anzahl der eingebauten Matche in G' durch $|V(G)|$ beschränkt ist und für jedes Match und jeden Input der Pfad des Inputs durch das Match in linearer Zeit bestimmt werden kann. □

Abbildung 15 veranschaulicht die gerade entwickelte Vorgehensweise. Der linke Graph entspricht hierbei G mit Ressourcenverbräuchen $y_1 ... y_4$ ausgewählter Kanten . Die Ressourcenverbräuche der restlichen Kanten ändern sich durch das dargestellte Match nicht. Der rechte Graph entspricht dem äquivalenten Graph und jeder der Kanten aus G' ist das entsprechende Delay aus $d_1 ... d_8$ zugeordnet.

Diese Routine wird im Folgenden Edge Resources genannt.
Wie bereits in Kapitel 3.2.1 erläutert, ist für die in dieser Arbeit verwendeten Technology Mapping Algorithmen das Klonen von Gates nicht erlaubt. Lockert man diese Vorgabe, bleibt es offen zu zeigen, wie Ressourcenverbräuche von neu enstandenen Gates und Kanten in die Aktualisierung der Verbräuche von G eingebracht werden.



$$y_1 \leftarrow y_1 + \frac{\xi \cdot (d_1 + d_4 + d_5)}{RAT}$$
$$y_2 \leftarrow y_2 + \frac{\xi \cdot (d_2 + d_6)}{RAT}$$
$$y_3 \leftarrow y_3 + \frac{\xi \cdot (d_3 + d_7)}{RAT}$$
$$y_4 \leftarrow y_4 + \frac{\xi \cdot d_8}{RAT}$$

Abbildung 15: Beispiel einer Aktualisierung von Ressourcenverbräuchen

5.2.4 Algorithmus

Im Folgenden wird ein Resource Sharing Algorithmus zur Lösung des Technology Mappings vorgestellt.

Algorithmus : Resource Sharing Algorithmus für das TM

Input : Instanz G des TM-Problems ohne teilweise überflüssige Subcircuits; ein Power-Budget B, $\gamma > 0, I \in \mathbb{N}$

Output : Konvexkombination von Kandidaten auf der TM-Instanz

1 $k[] \leftarrow \emptyset, \quad \phi[] \leftarrow \emptyset, \quad \Xi \leftarrow 0$
2 $y_e \leftarrow 0 \quad \forall e \in E(T_G), \quad y_{Power} \leftarrow 0$
3 **for** $i = 1, ..., I$ **do**
4 $\quad \omega_{Power} \leftarrow e^{\gamma \cdot y_{Power}}$
5 $\quad (\omega_e)_{e \in E(T_G)} \leftarrow Edge\_Weights(\gamma, (y_e)_{e \in E(T_G)})$
6 $\quad k[i] \leftarrow TM\_Oracle(\omega)$
7 $\quad \xi \leftarrow \min\{\frac{B}{Power(k[i])}, \frac{RAT_P}{Delay(P)} \forall P \in \mathcal{P}_{k[i]}\}$
8 $\quad$ **if** $\xi \geq 1$ **then**
9 $\quad\quad$ **return** $k[i]$
10 $\quad (y_e)_{e \in E(T_G)} \leftarrow Edge\_Resources(k[i], (y_e)_{e \in E(T_G)}, \xi)$
11 $\quad \phi[i] \leftarrow \xi, \quad \Xi \leftarrow \Xi + \xi$
12 **return** $k, \frac{\phi}{\Xi}$

Wenn der Algorithmus nicht in Schritt 9 endet, so nehme man den Kandidaten in k mit dem größten Faktor in ϕ/Ξ. Endet er in Schritt 12, so nehme man den besten gefundenen Kandidaten im Kandidatenvektor k.

5.2.5 Abschließende Bemerkungen

Im bisherigen Verlauf dieses Kapitels wurde B als gegeben vorausgesetzt. Es lässt sich jedoch B durch eine binäre Suche und durch einen Aufruf des Resource Sharing Algorithmus für jede Iteration der Suche ersetzen. Dies ergibt eine, wenn möglich RAT-einhaltende, Lösung mit möglichst kleinem Power-Verbrauch.
Des Weiteren lässt sich auch mit einem durch Technology Mapping umgebauten Circuit in der nächsten Iteration des Resource Sharing weiterrechnen, jedoch müssen dann die Preise und Ressourcen entsprechend angepasst werden. Dies ist für den gesamten Circuit nicht sinnvoll, da sonst alle Preise evtl. ihre Gültigkeit verlieren. Sehr kritische Subcircuits lassen sich jedoch mit entsprechenden Veränderungen an Ressourcen und Preisen austauschen. Diese kritischen Bereiche lassen sich auch mithilfe des Preprocessings verändern und in der nächsten Iteration separat bearbeiten. Die Kantenpreise können aufgrund der Flusserhaltung leicht in den neuen Graphen übertragen werden, jedoch ist zu diesem Zeitpunkt noch offen, wie die Summe der Ressourcenverbräuche aufgeteilt auf die Kanten in die neue Umgebung transformiert werden kann. Für Graphen mit polynomiell beschränkter Pfadmenge, beispielsweise Graphen mit konstant vielen Multifanoutknoten, lässt sich dies jedoch auch über die Pfadwerte direkt in

polynomieller Zeit ändern.
Die Definition der Ressourcen und Kunden wurde so allgemein gehalten, dass in einer Iteration des Resource Sharing Algorithmus das Technology Mapping sehr einfach durch eine andere die Logik optimierende Routine ausgetauscht werden kann. Abhängig von den aktuellen Preisen und den bisher errechneten Kandidaten ist es möglich, zum Beispiel das Technology Mapping Orakel mit dem in [DHHS18] beschriebenen Gate Sizing Algorithmus zu ersetzen. Dadurch ist eine deutlich höhere Flexibilität bei der Logik-Optimierung eines Ciruits möglich.
Ob der Resource Sharing Algorithmus für das Technology Mapping polynomiell implementierbar ist, hängt nur von dem gewählten Technology Mapping Algorithmus ab, denn alle anderen Schritte sind wie bereits in Kapitel 5.1.1 beschrieben, polynomiell implementierbar.

6 Qualitäts- und Laufzeit-Analyse

In diesem Kapitel werden die vorgestellten Heuristiken bezüglich Laufzeit und Güte analysiert. Vor dieser Analyse ist es jedoch notwendig, ein paar Angaben zur Struktur realer Instanzen zu machen, um die Ergebnisse der Algorithmen besser einordnen zu können.

Die im Folgenden behandelten Chips liegen mir aufgrund der Kooperation zwischen IBM und dem Forschungsinstitut für Diskrete Mathematik in Bonn vor. Auch die in dieser Arbeit vorgestellten Algorithmen wurden im Rahmen dieses Projektes implementiert. Die Instanzen dieser Chips werden als reale Instanzen bezeichnet.

6.1 Struktur realer Instanzen

Die Instanzen eines Chips sind alle maximal zusammenhängenden, mit Technology Mapping vollständig bearbeitbaren Circuits eines Chips. Da auf einem Chip beispielsweise durch Register gerichtete Kreise entstehen oder Bauteile existieren, welche nicht in der Library vorhanden sind, lässt sich der Logik-Graph eines Chips nicht vollständig mit dem Technology Mapping Algorithmus verarbeiten.
Das Chipdesign beinhaltet sehr viele Routinen, welche aus einem Bauplan einen produzierbaren Chip designen. Auf diesem Weg gibt es viele Zwischenstände (Snapshots). Alle getesteten Chips wurden auf dem Stand des selben Snapshots bearbeitet. Dadurch lassen sich die Verbesserungen von Instanzen durch die Algorithmen miteinander vergleichen, auch wenn sie von verschiedenen Chips stammen.
Die folgenden Angaben betrachten alle Instanzen zusammengesetzt zu einem Circuit. Insgesamt wurden 1107 Instanzen getestet.

Sei H_C die Menge Multifanoutknoten, welche keine Inputs sind, eines Circuits C und NI_C die Menge aller Knoten abzüglich der Input-Knoten von C. Der durchschnittliche Anteil von H_C an NI_C beträgt $30,95\%$. Die Menge der Multifanoutknoten, welche ebenfalls Inputs sind, wurde hier nicht beachtet, da diese nur einen möglichen Kandidaten besitzen und somit nicht zu

dem Problem des exponentiellen Klassenwachstums beitragen. Diese Werte besitzen bei den größeren Instanzen eine kleine Varianz. Aus diesem Grund werden die Circuits im Folgenden nur anhand ihrer Knotenzahl analysiert.

Der Anteil von Outputs an der Gesamtzahl von Knoten eines Circuits liegt bei $15,26\%$. Von allen Outputs haben $4,13\%$ noch weitere Nachfolger im gleichen Circuit. Outputs mit Nachfolgern werden ebenfalls auf einen Kandidaten beschränkt damit auf ihnen nicht zwei verschiedene Kandidaten implementiert werden, da sie zu den offenen Knoten darauffolgender Knoten gehören. Da diese nur einen solch geringen Anteil an allen Outputs besitzen, werden nur wenige Outputs so früh festgelegt. Auch hier wurden Outputs, welche auch Inputs sind, aus den gleichen Gründen wie oben nicht betrachtet.

$\lvert V(C)\rvert$	**#Instanzen**	**%**
>10	693	63%
>100	385	32%
>1000	189	17%
>10000	19	1,7%
>20000	12	1%
>30000	9	0,8%
>40000	4	0,04%

Abbildung 16: Größenverteilung der getesteten Instanzen

Die Inputs eines Circuits machen durchschnittlich $16,73\%$ an der Gesamtzahl der Knoten aus. Daraus folgt, dass etwa jeder achte Knoten nur einen Kandidaten trägt. Dies ist bei der Interpretation der folgenden Analysen zu beachten.
Abbildung 16 zeigt die Verteilung der getesteten Instanzen hinsichtlich der Größe ihrer Knotenmenge.

6.2 Analyse der Ergebnisse

6.2.1 Tradeoffparameter

Die Werte von Area, SNS, WS und des Energieverbrauchs eines Circuits liegen selten in derselben Größenordnung. Dies kann zur Folge haben, dass selbst ein vielfacher Verbrauch einer Ressource sich nicht in den Kosten des Circuits-Kandidaten bemerkbar macht. In der Praxis liegen Power, Area oder Arrivaltime häufig um mehrere Zehnerpotenzen auseinander.
Der Unterschied zwischen dem Verbrauch des Gates mit dem geringsten zu dem Gate mit dem höchsten Ressourcenverbrauch liegt bei maximal Faktor 10.
Der Ressourcenverbrauch eines Gates wird nun durch Dividieren durch den Verbrauch eines sparsamen Gates normalisiert; dann liegen alle Ressourcenverbräuche in derselben Größenordnung.
Im Folgenden wird ausgehend von der Technology Mapping Heuristik ohne Preprocessing und mit Premapping durch Schätzen der Einfluss des Tradeoffparameters λ auf die Änderung der Kosten untersucht. Die Kosten-

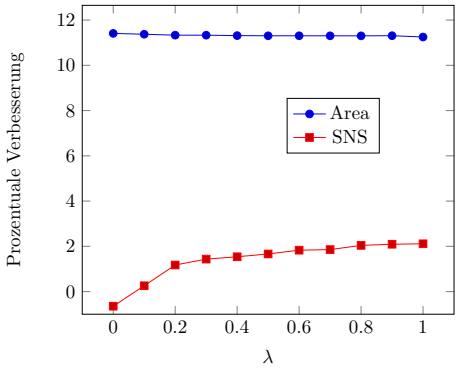


Abbildung 17: Einfluss von λ auf SNS und Area

funktion ist hierbei $\lambda SNS(C) + (1 - \lambda)Area(C)$.
Abbildung 17 veranschaulicht die durchschnittliche prozentuale Verbesserung der Kosten abhängig von λ.

Hier ist nur eine schwache Abhängigkeit bezüglich des Tradeoffparameters zu erkennen. Dies liegt daran, dass im Schnitt für jeden dritten Knoten ein Kandidat geschätzt wird. Des Weiteren liegen SNS und $Area$ oft in derselben Größenordnung, wodurch ein schnellerer Kandidat mit nahezu den gleichen Kosten geschätzt wird, wie ein kleinerer. Dadurch werden unabhängig des λ, sehr kostenähnliche Kandidaten gewählt; wodurch sich die Kosten des äquivalenten Circuits in Abhängigkeit des λ nur gering verändern.
Unabhängig von λ zeigt die Abbildung 17 jedoch auch ein enormes Verbesserungspotenzial des Circuits in SNS und besonders in $Area$.
In folgenden Tests wird aus diesem Grund mit konstantem $\lambda = \frac{1}{2}$ gerechnet.
Die Verbesserungen im vorgestellten FPTAS sind sehr stark durch λ beeinflusst. Eine ausführliche Analyse diese Algorithmus findet sich in [Elb17].

6.2.2 Premapping

Im Folgenden wird das triviale Premapping mit dem Premapping durch Schätzen verglichen. Dies geschieht in Abhängigkeit von der Größe der behandelten Instanzen.

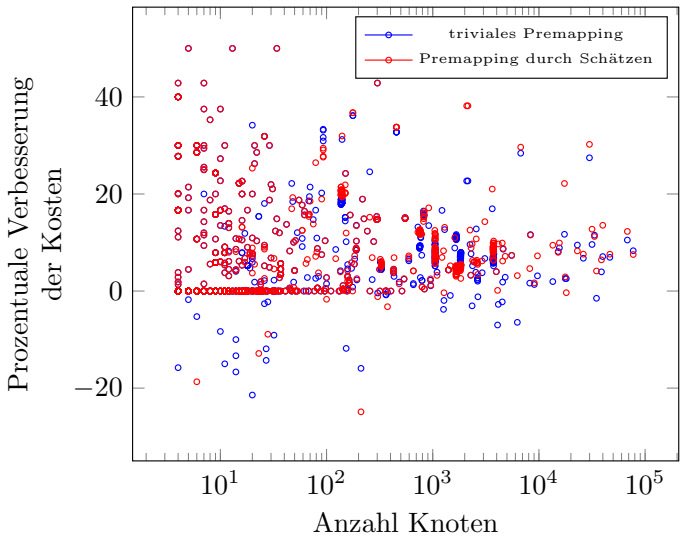


Abbildung 18: Vergleich der Kostenverbesserung von zwei Premapping-Methoden

Abbildung 18 bildet jeden mit der Heuristik gelösten Circuit C bezüglich der Anzahl seiner Knoten und der prozentualen Veränderung der Kosten von C $(\lambda \cdot SNS(C) + (1-\lambda) \cdot Area(C)$ ab. Jede Instanz wird einmal mithilfe des trivialen Premappings (blau) und einmal mit dem Premapping durch Schätzen (rot) gelöst.
Hierbei fällt auf, dass sich die Ergebnisse der beiden Premapping Methoden kaum unterscheiden, jedoch haben sich mit dem Premapping durch

Schätzen deutlich weniger Instanzen verschlechtert.

Im Durchschnitt wurden die Kosten der Instanzen durch das triviale Premapping (tP) um 6,74% und durch das Premapping durch Schätzen (PdS) um 6,46% verbessert. Aufgeteilt auf Area und SNS ergibt sich eine durchschnittliche Verbesserung durch das tP von 8,83% Area und 2,75% SNS und mit dem PdS von 10,42% Area und 2,16% SNS.

Abbildung 19 gibt einen Überblick über die Verteilung dieser Verbesserungen. Hierbei wird aufgezeigt, wie groß der Anteil der Instanzen ist, welche eine Verbesserung, entsprechend der in der ersten Spalte dargestellten Prozent-Bereiche, erzielt.
Aus diesen Gründen ist das Premapping durch Schätzen dem trivialen Premapping vorzuziehen. In den folgenden Tests wird deshalb das Premapping durch Schätzen als Premapping Methode genutzt.

%	tP	PdS
<0	3,12%	0,85%
$[0;5]$	51,56%	56,95%
$(5;10]$	19,68%	16,37%
$(10;20]$	16,65%	15,33%
$(20;30]$	4,92%	6,62%
>30	4,07%	3,88%

Abbildung 19: Verteilung der Kostenverbesserungen

6.2.3 Preprocessing

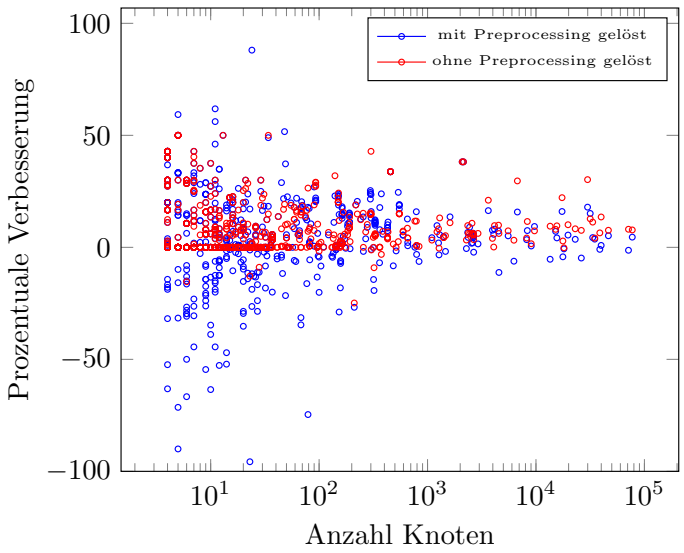


Abbildung 20: Kostenverbesserung in Abhängigkeit der Anzahl von Knoten der Instanzen

In diesem Kapitel wird die Heuristik mit und ohne Preprocessing verglichen. Abbildung 20 zeigt die prozentuale Verbesserung der Kosten der zugrundeliegenden Circuits in Abhängigkeit von der Anzahl der Knoten. Es ist deutlich zu sehen, dass durch das Preprocessing deutlich bessere Lösungen möglich sind. Jedoch wird die Ausgangsinstanz durch das Auseinanderbauen der Gates deutlich schlechter und der Algorithmus ist oftmals nicht in der Lage die Kosten auf das Niveu des Ausgangscircuits zu bringen. Dies lässt sich anhand der vielen Instanzen erkennen, die nach der Heuristik höhere

Kosten tragen.
Im Schnitt erhöhte sich die Anzahl der Knoten um 67%, die Instanzen wurden 39% größer und die SNS erhöhte sich um 16%. Aufgrund der Vielzahl von negativen Verbesserungen wird im Folgenden ohne Preprocessing weitergerechnet.
Das Preprocessing könnte anstatt auf den gesamten Circuit auch nur auf besonders kritische Bereiche angewandt werden. Es bleibt offen zu zeigen, wie solche Subcircuits zu finden sind und wie sich anhand dieser das Preprocessing optimal einsetzen lässt.

6.2.4 Gütevergleich mit kleinen optimal gelösten Instanzen

In diesem Kapitel werden kleine Instanzen mit einem Output betrachtet. Es handelt sich um die für den FPTAS geeigneten Instanzen, welche in Kapitel 4.1.1 beschrieben wurden, mit zufällige Arrivaltimes der Inputs. Hierbei wurden jeweils zwei Instanzen mit 5-150 Input-Knoten generiert. Die RAT des Outputs wurde auf 0 gesetzt.
Jede Instanz wurde einmal mit der Heuristik ohne Preprocessing und mit dem Premapping durch Schätzen (hellrot) gelöst. Ein weiteres Mal wurden diese Instanzen mithilfe des Technology Mappings, welches jeden nicht dominierten Kandidaten an jedem Knoten speichert und das Filtern mit Buckets nicht ausfürt, gelöst (blau). Dies führt zu einem nicht polynomiellen Algorithmus, garantiert jedoch eine kostenoptimale Lösung. Abbildung 21 stellt diese beiden Routinen bezüglich der prozentualen Kostenverbesserung abhängig von der Anzahl der Knoten jeder Instanz dar. Die Instanzen, bei denen beide Routinen zu demselben Ergebnis gekommen sind, wurden dunkelrot markiert.
Es fällt auf, dass die Verbesserungen der optimal gelösten Instanzen und der durch die Heuristik gelösten Instanzen im Schnitt sehr nah beinander liegen. Im Durchschnitt ist die optimale Routine um $0,968\%$ Prozent besser als die Heuristik. Dieses Resultat lässt sich jedoch nicht unmittelbar auf die Instanzen eines Chips übertragen, denn der Anteil an Multifanoutknoten liegt hier nur bei $2,67\%$, statt bei $30,95\%$ wie auf den Instanzen eines Chips. Zu beachten ist außerdem, dass daraus nicht abgeleitet werden kann, wie hoch die Kosten einer optimalen Realisierung der dem Circuit zugrunde liegenden booleschen Funktion ist.

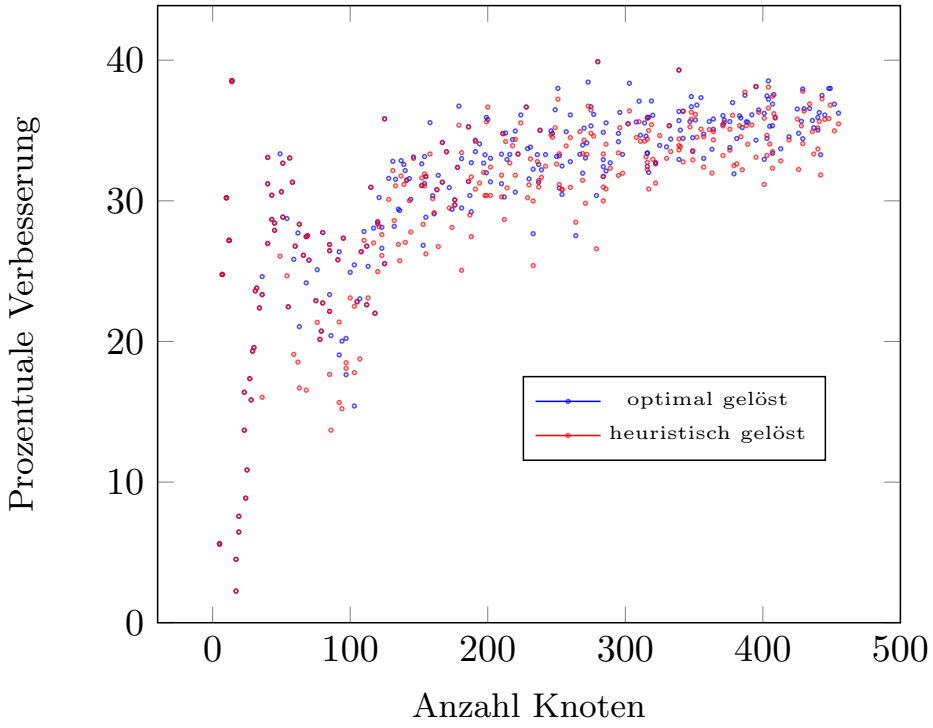


Abbildung 21: Vergleich der Kostenverbesserung kleiner heuristisch und optimal gelöster Instanzen

6.3 Laufzeitanalyse

Aus der theoretischen Laufzeitschranke aus Kapitel 3.8 folgt, dass die Laufzeit bei beliebig vielen Multifanoutknoten für die Match-Suche nicht polynomiell beschränkt ist.
Die vorangegangenen Testkapitel haben folgendes Ergebnis hervorgebracht: Die Technology Mapping Heuristik mit dem Premapping durch Schätzen und ohne Preprocessing führen auf den getesteten Instanzen zu den besten Ergebnissen.

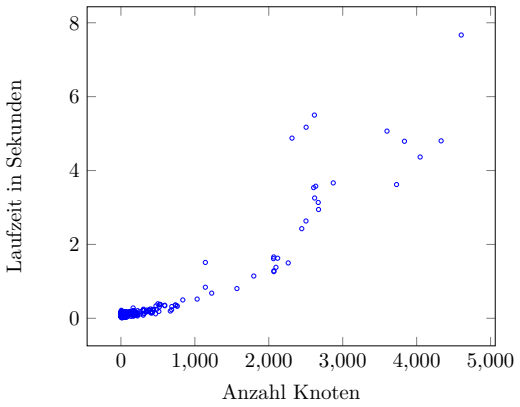


Abbildung 22: Übersicht über die Laufzeit der Heuristik auf allen Instanzen mit maximal 5000 Knoten

Die Abbildungen 22 und 23 geben einen Überblick über die Laufzeit der Heuristik mit gerade genannten Einstellungen in Abhängigkeit von der Anzahl der Knoten der Instanzen. Hierbei gibt Abbildung 23 einen Überblick über alle Instanzen und somit eine globale Laufzeitanalyse und Abbildung 22 betrachtet alle Instanzen mit maximal 5000 Knoten.
Auf beiden Abbildungen lässt sich der exponentielle Anstieg der Laufzeit nicht eindeutig erkennen, da nur sehr wenige Instanzen mit mehr als 30.000 Knoten vorhanden sind. Abgesehen von der größten Instanz ließen sich jedoch alle Instanzen in deutlich unter einer Stunde lösen.

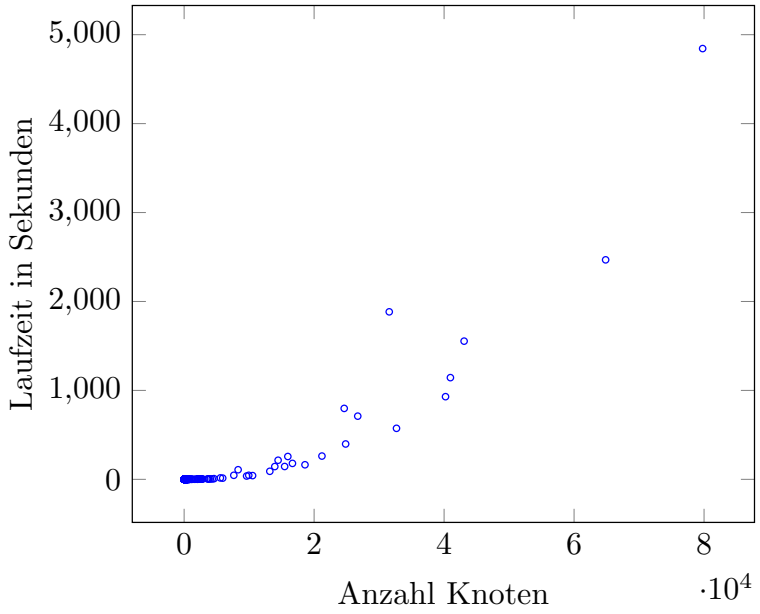


Abbildung 23: Übersicht über die Laufzeit der Heuristik auf allen Instanzen

6.4 Einzelbetrachtung zweier Chips

In diesem Kapitel werden zwei Chips (Chip1, Chip2) gesondert analysiert.Chip1 ist ein aktueller 7nm (Nanometer) Chip und besitzt die volle in Abbildung 2 angegebene Library. Chip2 hingegen besitzt eine kleinere Library ohne MUX(I), (N)ANDN, NOR4, OR und AND Gates.

Auf Chip1 liegen 5 Instanzen mit einer durchschnittlichen Größe von 813 Knoten und Chip2 besitzt 96 Instanzen mit einer durchschnittlichen Größe von 105 Knoten.

Eigenschaft	Chip1	Chip2
SNS	$4,4\%$	0%
Area	$6,0\%$	$12,6\%$

Abbildung 24: Verbesserung der Instanzen auf Chip1 und Chip2

Abbildung 24 listet die für das Technology Mapping wichtigen Eigenschaften des Chips, vor und nach der Bearbeitung mit der im vorangegangenen Kapitel beschriebenen Heuristik, auf. Für die in dieser Abbildung aufgezeigten Verbesserungen durch die Heuristik, benötigte diese auf Chip1 eine Laufzeit von 4,0 Sekunden und auf Chip2 wurden 4,6 Sekunden benötigt.
Die gegebenene Werte für Delay und Area sind eine Schätzung der realen Werte. Daraus folgt, dass sich aus den positiven Verbesserungen bezüglich dieser Schätzungen nicht auf reale Verbesserungen schließen lässt.

Eigenschaft	Chip1	Chip2
SNS	$28,7\%$	-0.06%
Area	$-1,9\%$	$0,01\%\%$
WS	$2,5\%$	0%
Netzlänge	$-11,3\%$	$-6,4\%$

Abbildung 25: Globale Verbesserungen auf Chip1 und Chip2

Die Tests auf den Chips zeigen, dass die vermutete und reale Verbesserung durchaus weit auseinander liegen können und oft Area oder SNS des Chips schlechter wird als vor der Bearbeitung. Dies veranschaulicht Abbildung 25, welche die realen Veränderungen auf den beiden Chips nach Anwendung der Heurstik aufzeigt. Es bleibt offen

zu zeigen wie die Schätzungen so genau gewählt werden können, dass sich die deutlichen Verbesserungsmöglichkeiten durch Technology Mapping auch auf die realen Werte eines Chips übertragen lassen.

7 Fazit und Ausblick

Das Ziel dieser Arbeit war es den von [Elb17] entwickelten Ansatz einer Heuristik auf meherere Outputs, RATs und eine beliebige Library zu erweitern und zu implementieren und einen Ansatz für das Technology Mapping mit Resource Sharing zu entwickeln.
Dies wurde mit folgendem Ergebnis erledigt: Der Heuristik ist es möglich in kurzer Zeit auf mehreren 10.000 Gates großen in vielen Fällen eine Verbesserung der Kosten, bestehend aus Area und SNS, zu erzielen. Hierbei wurden die besten Ergebnisse mit dem Premapping durch Schätzen und ohne das Preprocessing erreicht und festgestellt, dass der Tradeoffparameter keinen großen Einfluß die Güte dieser Ergebnisse hat.
Des Weiteren wurde ein Ansatz zur Lösung des Technology Mapping mithilfe des Resource Sharing Algorithmus entwickelt. Für diesen Ansatz steht die Implementierung jedoch noch aus.

Zukünftig gilt es den Ansatz mithilfe des Resource Sharing zu implementieren und zu testen. Des Weiteren müsste gezeigt werden wie das Technology Mapping im Resource Sharing mit anderen die Logik optimierenden Routinen (z.B. Gate Sizing) kombiniert werden kann um möglichst kosteneffieziente Lösungen zu garantieren.
Mit der Hinzunahme mehrerer Implementierungen von Gates in die Library, ließe sich ebenfalls eine Lösung des Gate Sizing Problems approximieren.
Es bleibt offen zu zeigen ob durch das erweiterte Premapping durch Schätzen eine verlässliche Verbesserung gegenüber dem Premapping durch Schätzen erzielt werden kann.
Des Weiteren könnten weitere Optimierungskriterien, wie die Netzlänge oder die Power, zusätzlich mit in die Kostenfunktion des Technology Mapping eingebracht werden um übermäßigen Verbrauch von im Technology Mapping nicht beachteten Ressourcen zu verhindern.

8 Literaturverzeichnis

[BH18] U. Brenner and A. Hermann. Faster Carry Bit Computation for Adder Circuits with Prescribed Arrival Times. *Research Institute for Discrete Mathematics, University of Bonn*, 2018.

[Dab17] S. Daboul. Algorithms for the gate sizing and Vt assignment problem. *Forschungsinstitut für Diskrete Mathematik Bonn*, 2017.

[DHHS18] S. Daboul, N. Hähnle, S. Held, and U. Schorr. Provably Fast and Near-Optimum Gate Sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, February 2018.

[Elb17] L. Elbert. Approximationsalgorithmen für das Technology-Mapping. *Forschungsinstitut für Diskrete Mathematik Bonn*, 2017.

[Häh15] N. Hähnle. Time-Cost Tradeoff and Steiner Tree Packing with Multiplikative Weights. *Technical report no. 1511115, Research Institute for Discrete Mathematics, University of Bonn*, 2015.

[HH15] S. Held and J. Hu. Gate Sizing. In *Electronic Design Automation for Integrated Circuits Handbook*, 2015.

[HS17] S. Held and S. Spirkl. Fast Prefix Adders for Non-Uniform Input Arrival Times. *Algorithmica*, 2017.

[Keu87] K. Keutzer. DAGON: Technology Binding and Local Optimization by DAG Matching. In *24th ACM/IEEE Design Automation Conference*, pages 341–347, June 1987.

[KHF18] H. N. Khan, D. A. Hounshell, and E. R. H. Fuchs. Science and research policy at the end of Moore's law. *Nature Electronics*, 1, January 2018.

[KR89] K. Keutzer and D. Richards. Computational Complexity of Logic Synthesis and Optimization. In *International Workshop on Logic Synthesis*, 1989.

[MCB+04] A. Mishchenko, S. Chatterjee, R. Brayton, X. Wang, and T. Kam. Technology Mapping with Boolean Matching, Supergates and Choices. 2004.

[Moo65] G. E. Moore. Cramming More Components onto Integrated Circuits. *Electronics Magazine*, 38, April 1965.

[MRV11] D. Müller, K. Radke, and J. Vygen. Faster min–max resource sharing in theory and practice. *Mathematical Programming Computation*, 3(1):1–35, Mar 2011.

[Pos41] E. L. Post. The Two-Valued Iterative Systems of Mathematical Logic. *Annals of Mathematics*, 5, 1941.

[Tra15] K. Van Tran. Algorithmen für das Technology-Mapping. *Forschungsinstitut für Diskrete Mathematik Bonn*, 2015.

[Wer07] J. Werber. Logic Restructuring for Timing Optimization by Restructuring Long Combinatorial Paths. *Research Institute for Discrete Mathematics, University of Bonn*, 2007.

[WRS07] J. Werber, D. Rautenbach, and C. Szegedy. Timing Optimization by Restructuring Long Combinatorial Paths. *IEEE/ACM International Conference on Computer-Aided Design*, 2007.