Algorithmen für das Technology Mapping

Alexander Zorn

Geboren am 26. Mai 1996 in Bonn

5. 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 [HNKF18] wird dies jedoch in absehbarer Zeit nicht mehr möglich sein, was das folgende Thema weiter in den Vordergrund rückt.
Derzeit beträgt die Dichte an Transistoren, die zu einem integrierten Schaltkreis auf einem Chip miteinander verbunden sind, 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 beweist.

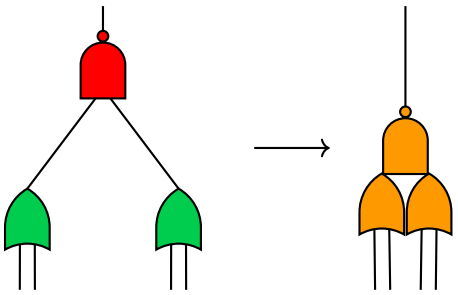

Abbildung 1: Zwei Realisierungen der logischen Funktion $\neg((w \vee x) \wedge (y \vee z))$

Die Größe der Menge aller möglichen Baupläne (später Circuits) für eine logische Funktion hängt maßgeblich von der auf dem Chip zur Verfügung stehenden Bauteilen, sowie von dem Aufbau der Funktion, ab. Hierdurch bedingt ergeben sich eine Vielzahl Möglichkeiten eine logische Funktion zu realisieren.
Jedes Bauteil hat spezifische physikalische Eigenschaften an Größe, Geschwindigkeit (Delay) etc.. Demnach hat auch jeder Circuit entsprechende Eigenschaften.
Ziel des Technology Mappings ist es, für eine 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, 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 kleinen oder eingeschränkten gegebenen Bauplänen noch in kurzer Zeit finden. Die Lösung dieses Pro-

blems für allgemeine Baupläne und Kostenfunktionen ist jedoch ein NP-vollständiges Problem. Aus diesem Grund entwickelt die folgende Arbeit eine Heuristik, welche für sehr (mehrere 10.000 Bauteile) große Baupläne in möglichst kurzer Zeit einen kostenoptimierten Umbau ermöglicht.
am ende hier noch eine kurze Quellenübersicht geben an lukas orientiert

2 Terminologie und grundlegender Algorithmus

2.1 Grundlegende Definitionen

Es folgen ein paar grundlegende Definitionen für die Beschreibung des Problems.

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 $f_g : \{0,1\}^n \to \{0,1\}$.
Eine Library L ist eine Menge von Gates und sei
$fanin_{max} := max\{arity(g) | g \in L\}$.

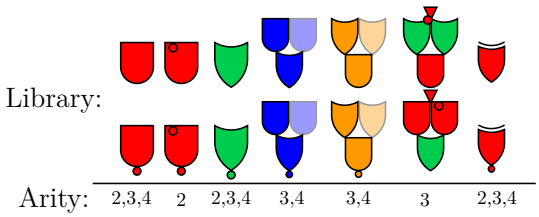

Abbildung 2: Beispiel einer Library

$area_g$ 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 $d_g \in \mathbb{R}^n$ wählt und somit Zeiten für jeden der Inputs angeben werden können. Hierauf wird jedoch erst in Kapitel 7 eingegangen.
Wenn die Signale der Inputs nicht zur selben Zeit ankommen wird, falls nicht anderes angegeben, gewartet bis das letzte Signal das Gate erreicht.

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

- **Inputs** mit Eingangsgrad Null.
- **Gates** mit mindestens einer eingehenden und einer ausgehenden Kante. Diese Knoten entsprechen der Gate Definition 2.2 und darüber hinaus gilt: An jeder eingehenden Kanten kann ein Inverter liegen.

- **Outputs** mit genau einer eingehenden Kante und keiner ausgehenden.

Ein Gate mit mehr als einer ausgehenden Kante wird auch Highfanoutgate 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 einem Circuit lassen sich Teilgraphen durch ein Gate der Library austauschen. Voraussetzung für einen solchen Tausch ist, dass der veränderte Circuit äquivalent zu dem Original ist. Dies wird in den folgenden Definitionen formalisiert.

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

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

So dass der Circuit C', welcher durch den Austausch des Subcircuits von X bis g durch das Match (mit den durch inv 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 (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 Outputknoten-Menge O. Eine Circuit-Kandidat K von C ist eine Menge von Kandidaten, so dass $\forall o \in O \, \exists! h \in K : h$ ist Kandidat von o und an jedem Knoten von C an dem sich mehrere Kandidaten überschneiden ist dasselbe Match gewählt.

Abbildung 3 visualisiert die vorherigen Definitionen.

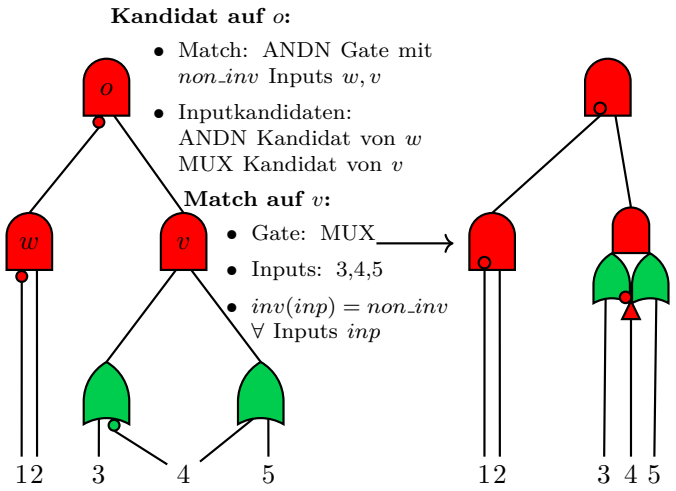

Abbildung 3: Beispiel eines Kandidaten und eines Matches

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 besten Circuit-Kandidaten auf C zu finden. Dafür ist eine Maß notwendig, durch welche diese Kandidaten bewertet werden. Es folgen zwei praxisrelevante Beispiele einer solchen Funktion. In der Praxis und in dieser Arbeit wird hauptsächlich eine konvex-Kombination aus beiden verwendet.

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

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

Wobei $can(K)$ die Menge der Kandidaten von K ist, $area_{inv}$ die Größe eines Inverters und $inputs(k)$ die Inputknoten des Outputknoten des Kandidaten k sind. Des Weiteren ist d_i das Delay eines Inverters und $d_{w(k,i)}$ das Delay der Kante zwischen den Knoten k und i. $inp\_can(k,i)$ gibt den Kandidaten des i'ten Inputs von k zurück.

Das Delay (AT) gibt an, wann das letzte Signal aus einem der Outputs des Circuit kommt.

2.2 Kern Algorithmus

Es folgt ein grundlegender Algorithmus, welcher 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 Highfanoutknoten (Knoten mit nur einer ausgehenden Kante), 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 finalem Output o, Library L

```
1 bester_kandidat[] ← ∅
2 bester_inv_kandidat[]← ∅
3 foreach Knoten v ∈ V(G) in topologischer Reihenfolge do
4     berechne alle (invertierten) Matches auf v
5     foreach Match m auf v do
6         Berechne besten Kandidaten mit m auf v
7         Update best_(inv)_kandidaten
8 Implementiere C entsprechend bester_kandidat[o]
```

Ein Technology Mapping Algorithmus liefert in der Regel nicht die bestmögliche Implementierung der C zugrunde liegenden logischen Funktion. Dies veranschaulicht Abbildung 4. In diesem Beispiel ist die zugrundeliegende Funktion konstant und somit könnte man auf alle Gates verzichten. Dies ist jedoch mit den bisher eingeführten Möglichkeiten des Technology Mapping nicht möglich.

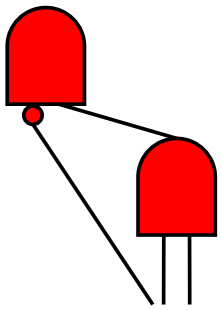

Abbildung 4: Ein Circuit desen boolesche Funktion $f = 0$ ist

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

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 Highfanoutknoten 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, so wird, wie soeben dargestellt, auch für v der schnellste bzw. kleinste Kandidat k gespeichert.
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'. k ist jedoch 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 Highfanoutgates, 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. Schritt 3 und 5 sind zwei verschachtelte Schleifen mit $|V(C)|$ und max $|L|$ Durchläufen.
Daraus folgt eine Laufzeit von $\mathcal{O}(|V(C)|^3|L|^2)$. □

3 FPTAS und Heuristik

In diesem Kapitel wird ein Approximationsalgorithmus für allgemeinere Instanzen und Zielfunktionen des Technology Mapping vorgestellt. Da auch dieser noch einige Einschränkungen vorgibt, welche für reale Instanzen eines Chips nicht gelten und das exakte Problem bereits NP-vollständig ist, wird auf Basis des Approximationsalgorithmus eine Heuristik entworfen.

3.1 Tradeoffprobleme

Der oben 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 Nachfrage 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, werden diese Problemstellungen auf Circuits mit mehreren Outputs erweitert.

Beide Probleme sind gleich schwer und 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 späteren Verlauf dieser Arbeit die Arrivaltime nicht der Einzige zu

beachtende Faktor bei der Geschwindigkeitsoptimierung ist, wird im Folgenden nur noch die Konvexkombination behandelt.

Es ergibt sich folgendes Problem für die Implementierung eines Algorithmus:

Angenommen an jedem Knoten v würde, wie im Kern-Algorithmus, nur derjenige Kandidat gespeichert werden, welcher die Kostenfunktion an v optimiert. Wenn dies der Fall ist kann nicht mehr garantiert werden, dass beim Errechnen der Kandidaten für den Output, der für ihn optimale Kandidat noch vorhanden ist. Beide Inputs getrennt nach der Kostenfunktion zu optimieren, garantiert also nicht das optimale Ergebnis.

Die Kosten eines Kandidaten k sind somit 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 muss; Hierzu folgende Definition.

Definition 3.1. (dominierte Kandidaten)
Seien k_1, k_2 Kandidaten desselben Knotens. Dann wird k_1 von k_2 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 Mapping verwendet nur nicht-dominierte Kandidaten, anderweitig ließe sich, durch ersetzen eines solchen durch einen nicht-dominierten eine bessere Lösung erzielen, was ein Widerspruch ist. Daraus folgt, dass nur diese während der Ausführung des Algorithmus gespeichert werden müssen.
Die Menge der noch bleibenden Kandidaten lassen sich in sogenannten Tradeoff-Kurven speichern (s. Abb. 5). Welche jeden Kandidaten zweidimensional anhand seiner Kosten erfasst.

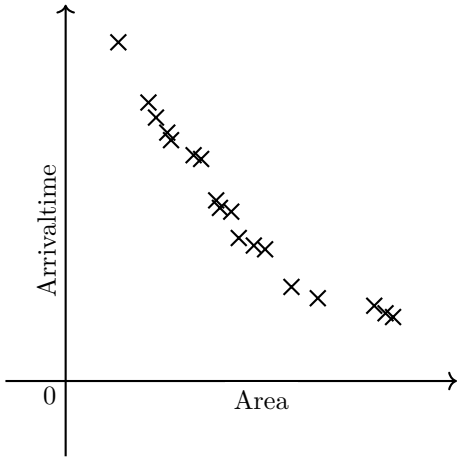

Abbildung 5: Ordnen der Kandidaten in einer Tradeoffkurve

Die beiden vorgestellten Probleme sind NP-vollständig. Daraus folgt, dass sich ab diesem Punkt wahrscheinlich kein polynomieller Algorithmus für

das Technology Mapping finden lässt, welcher, bezüglich der vorgestellten Operationen des Technology Mapping , den kostengünstigsten Circuit-Kandidaten liefert.
Da bei zwei nicht dominierenden Kandidaten nicht eindeutig ist welcher der bessere ist, wird eine Vielzahl von Kandidaten an jedem Knoten gespeichert. Dies zeigt sich in einem exponentiell großen Speicherbedarf.
Ein Beweis der NP-vollständigkeit und genauere Informationen zur Einordnung der Schwere dieser Probleme finden sich in [KR89] und [Elb17].

3.2 Highfanoutknoten

Der beschriebene Kern Algorithmus arbeitet nur auf Circuits, in denen keine Highfanoutknoten existieren. Dies kommt auf einem realen Chip jedoch sehr häufig vor; ca. 30% der Knotenmenge sind Highfanoutknoten. Eine Analyse über den Zusammenhang von der Anzahl der Highfanoutknoten und der Laufzeit des Algorithmus befindet sich Kapitel 9.

Es ist möglich, einen Circuit, in kleinere Subcircuits zu unterteilen, welche solche Highfanoutknoten 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 stellt diesen Ansatz einer Heuristik dar. Der Anteil an Highfanoutknoten ist auf den mir vorliegenden Chips so groß, dass eine Vielzahl sehr kleiner Subcircuits entsteht, woraus folgt, dass die Möglichkeiten des Technology Mapping sehr eingeschränkt werden. Aus diesem Grund werde ich diesen Weg einer Heuristik nicht weiter verfolgen.

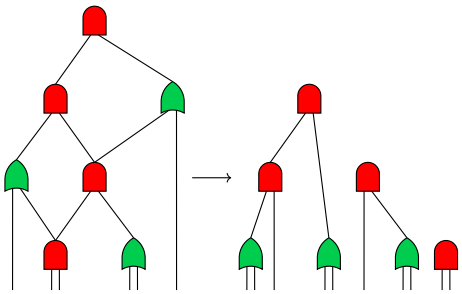

Abbildung 6: Unterteilen eines Circuit in Highfanoutfreie Subcircuits

Bei der Implementierung von Highfanoutknoten besteht die größte Herausforderung darin, dass bei der Konstruktion des äquivalenten Circuits die eingebauten Kandidaten aller Nachfolger eines Highfanoutknoten v an v übereinstimmen müssen. Daraus folgt, dass bei der Wahl eines Kandidaten für einen

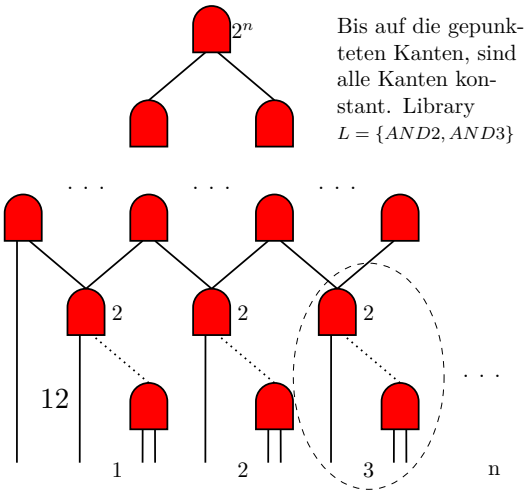

Abbildung 7: Exponentiell viele Kandidaten bereits bei sehr eingeschränkter Library

Knoten w die Wahl der Kandidaten der Input-Kandidaten von w nicht unabhängig von einander sein muss.
Abbildung 7 zeigt zudem ein weitere Herausforderung für die Implementierung auf. Die Anzahl der zu speichernden Kandidaten kann exponentiell bezüglich $|V(C)|$ sein.
Zur Lösung des ersten Problems 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 w offener Knoten von v, wenn folgendes gilt:

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

Somit ist die Menge der offenen Knoten eines Circuit Knoten v, die Menge aller Highfanoutknoten w, 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 Highfanoutknoten, welche durch die Kandidaten eines Knoten außerhalb von $cone(v)$ verändert werden können. Alle Kandidaten von Knoten mit Ausgangsgrad 1 und dieser Eigenschaft, sind durch den Nachfolger-Kandidaten (welcher auch zu einem offenen Knoten gehören muss), bereits eindeutig definiert.
Abbildung 8 visualisiert die vorangegangenen Definitionen.

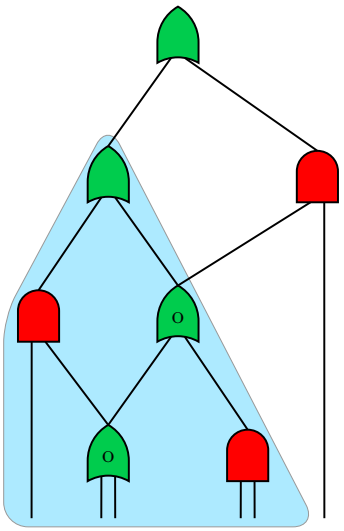

Abbildung 8: Visualisierung der Definitionen 3.2 und 3.3

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 Knoten v mit $class(k_1) \neq class(k_2)$ nicht miteinander vergleichen. Dies gilt auch für den Fall, 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 Highfanoutknoten 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

Es wird aus folgendem Grund angenommen, dass die Inputkandidaten eines Kandidaten an deren offenen Knoten übereinstimmen müssen.
Angenommen dies ist nicht der Fall dann werden 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 zu vermeiden ist. Der erhöhte Verbrauch von Kanten bringt höhere Routing Kosten mit sich, welche im Technology Mapping jedoch nicht berücksichtigt werden. Um zu verhindern, dass nicht beachtete Ressourcen übermäßig verbraucht werden, ist das Klonen in den vorgestellten Algorithmen nicht erlaubt und wird durch die Klassen und die Routine, welche beim Verknüpfen von Input-Kandidaten zu einem neuen Kandidaten genutzt wird, 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.
In Kapitel 4.2 wird eine zusätzliche Klasse von Kanten eingeführt, über welche man nicht matchen darf. Diese Kanten bezeichnet man als konstant. Deshalb wird bei der Bildung eines jeden Matches 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, dass die zusätzlichen Routing-Kosten unerwarteterweise vernachlässigbar sind und ein Match über diese Kante für eine optimale Lösung notwendig ist.

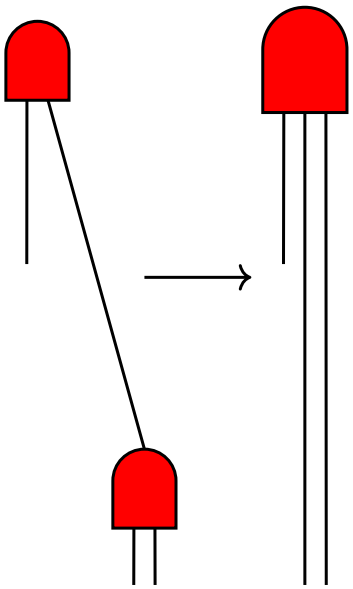

Abbildung 9: Visualisierung eines Matches über eine lange Kante

3.4 Teilweise überflüssige Subcircuits

Abbildung 4 beweist, 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 eine Teilmenge der Inputs gibt, von denen die Signale der Outputs nicht abhängen.

Die Berücksichtigung von vollständig überflüssigen Subcircuits würde bedeuten, dass Teile des Circuits entfernt werden und die Outputs der Inputs

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, jedoch birgt es ebenfalls einen großen Aufwand zur Implementierung in die aktuelle Architektur des Technology Mapping Algorithmus. 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, kommen auch in der Implementierung noch folgender Algorithmen nicht vor.

Im Gegensatz dazu kommen die teilweise überflüssigen Circuits sehr wohl 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.
Es gibt dabei jedoch noch eine Besonderheit. Wenn alle Inputs bis auf einen irrelevant sind, so 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 im implementieren. Diese zusätzliche Bearbeitung läuft dann in jedem folgenden Algorithmus automatisch im Hintergrund und findet keine Erwähnung mehr. oder doch ?
Obwohl nun das Gegenbeispiel aus Kapitel 2.2 nicht mehr gültig ist, lässt sich mit den Möglichkeiten des Technology Mapping in der Regel keine optimale Realisierung der des Circuits zugrunden liegenden logischen Funktion finden.

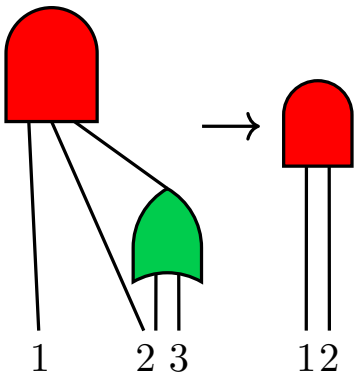

Abbildung 10: Nur Input 1 und 2 sind die relevanten Inputs

3.5 Matchingprobleme

In Circuits mit beliebig vielen Highfanoutknoten steigt die Anzahl der möglichen Matche eines Knoten sehr stark an, was eine direkte Auswirkung auf die Größe der Menge der möglichen Kandidaten hat.
Es folgt eine abstraktere Betrachtung von Matchings und daraus ableitend ein Algorithmus der auf realen Instanzen fast alle möglichen Matche findet und in polynomieller Zeit implementierbar ist.

3.5.1 Obere Schranke für die Anzahl Matche eines Knoten

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

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

Beweis. Zu jedem Inputpin eines Matches gehört eindeutig eine Kantemenge K aus C. Es ist möglich, dass $|K| > 1$, denn ein Match kann mehrere ausgehende Kanten eines Knoten 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 zu einem der Inputs des Matches gehören, denn sonst hätte das Match einen Seiteninput, was nicht erlaubt ist, oder es existiert ein teilweise überflüssiger Subcircuit. Daraus folgt die Aussage. □

Die Korrespondenz ist nicht eindeutig, denn, ob mehrere Kanten eines Outputs zusammengefasste oder getrennte Inputkanten eines Matches sind, lässt sich anhand des Schnittes nicht herleiten. Dies wird ebenfalls durch Abbildung 11 veranschaulicht. Ohne Highfanoutknoten ist diese Zuweisung jedoch, abgesehen von den möglichen Invertierungen der Inputs und des Outputs, eindeutig, wenn es

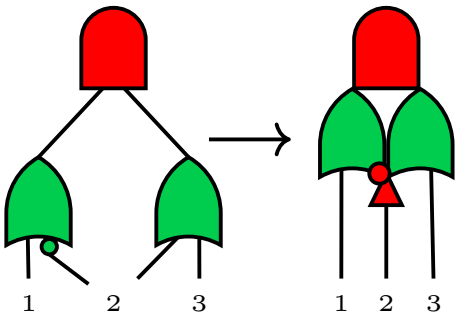

Abbildung 11: Das MUX Match verbindet zwei Kanten eines Highfanoutknoten

keine zwei Gates der Library gibt, welche diegleiche logische Funktion realisieren.
In einem Graphen ohne Highfanoutknoten gilt somit:
Die Menge der Matchings 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 interpretiert ein Gate der Library realisieren.

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

$$|X_v| = \sum_{i \leq fanin_{max}} \binom{n}{i} \leq \sum_{i \leq fanin_{max}} \frac{n}{i!(n - fanin_{max})!} \leq n^{fanin_{max}} fanin_{max}$$

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. Daraus folgt die Aussage. □

Für allgemeinere Circuits lässt sich folgende Schranke angeben:

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

Beweis. Korollar 3.7 gibt eine obere Schranke für die Anzahl aller Matche inklusive der möglichen Invertierungen an. Sei $u \in x \in X_v$ (Definition siehe oben) ausgehende Kante eines Highfanoutknoten 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 Highfanoutknoten lässt sich die Schranke noch genauer angeben.

Korollar 3.9. *Sei C ein Circuit ohne Highfanoutknoten. 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 Highfanoutknoten enthält, wird der Fanin des Matches um mindestens eins erhöht, denn es ist nicht möglich, Kreise zu schließen. Daraus folgt, dass der obige Schritt maximal $fanin_{max}$ mal durchführbar ist. Für jeden so berechneten Prototyp einen Matches gibt es noch $2^{fanin_{max}}$ mögliche Invertierungen der Inputs und 2 des Outputs, woraus die obige Aussage folgt. □

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 Knoten in einem Graphen mit beschränktem $fanin_{max}$ und $fanout_{max}$.

3.5.2 Matching Suche in polynomieller Zeit

Der Kernalgorithmus aus Kapitel 2.2 findet alle möglichen Matche eines beliebigen Knotens in polynomieller Zeit. Dies ist dort möglich, da in dem Circuit keine Highfanoutknoten 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 wir 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 Highfanoutknoten 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 Matching Suche

In der Praxis, ähnlich dem oben beschriebenen Fall ohne Highfanoutknoten, wird von dem Gate eines Knoten ausgehend überprüft ob dieser, oder eine Mögliche Invertierung, einem Gate der Library entspricht. Daraufhin wird jede der maximal $2^{fanin_{max}}$ Möglichkeiten die an den Input liegenden Gates mit in das potenzielle Match hinzuzufügen inclusive möglicher Invertierungen überprüft. Dieser Vorgang wird für jede der Möglichkeiten so lange wiederholt, bis die Anzahl der Inputs des durch das potenzielle Match, beschriebenen Subcircuit größer als $fanin_{max}$ ist. Ab diesem Punkt wird das vorliegende potenzielle Match nicht mehr erweitert.
Dies garantiert, bei einem Circuit mit Highfanoutknoten, nicht das Finden

aller möglicher Matche, da sobald ein potenzielles Match einen Highfanoutknoten mit einschließt die Zahl der Inputs sinken kann.
In der Praxis werden so jedoch die überwiegende Mehrheit der Matche gefunden. Der Verlust einiger weniger Matche spiegelt bringt einen enormen Laufzeitgewinn mit sich. die dissertation für den fall ohne highfanoutknoten erwähnen ? und nochmal genau durchsehen ob die teilweise überflüssigen Subcircuits überall rausgehalten wurden
boolean Matching hier erwähnen und sinnvolle quellen mit einbringen

3.6 Kandidaten-Probleme

Die Anzahl der Kandidaten an einem Knoten ist im Allgemeinen nicht polynomiell beschränkt, wie Abbildung 7 beweist. Dies gilt offenbar auch wenn der Circuit C keine Highfanoutknoten besitzt. Dieses Problem wird durch das Filtern mit Buckets gelöst. Dies wird nach dem Folgenden Korollar eingeführt.

Korollar 3.11. *Sei C ein Circuit und $v \in C$. Die Anzahl der Klassen von v ist exponentiell in $k := |offene\_Knoten(v)|$ beschränkt.*

Beweis. Sei K die größte Kardinalität einer Kandidatenmenge von einem Knoten $w \in offene\_Knoten(v)$. Jede Kombination von Kandidaten der offenen Knoten entspricht, solange sich diese nicht gegenseitig ausschließen, einer Klasse von v. Die Anzahl dieser Kombinationen ist durch K^k beschränkt. Daraus folgt die Aussage. □

3.6.1 Filtern mit Buckets

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.
Man wählt die Bucketgrößen $\delta_{delay} = \frac{\varepsilon}{2\lambda}, \delta_{area} = \frac{\varepsilon}{2(1-\lambda)}$ (mit Tradeoff λ), und speichert erst nur den kleinsten Kandidaten in jeder Area-Bucket und von den verbleibenden nur den schnellsten in jeder Delay-Bucket.
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. Mit anderen Worten wir im nicht definierten Fall die Tradeoffkurve in nur ein Bucket unterteilt.
Die maximale Anzahl an Kandidaten in einer Tradeoffkurve ist nach dem Filtern polynomiell beschränkt.

Definition 3.12. Sei $\mathcal{B}$ die maximale Anzahl an Kandidaten in einer Tradeoffkurve, nachdem sie gefiltert wurde.
Des Weiteren sei $\mathcal{K}$ die maximale Kardinalität von Klassen eines Knoten.

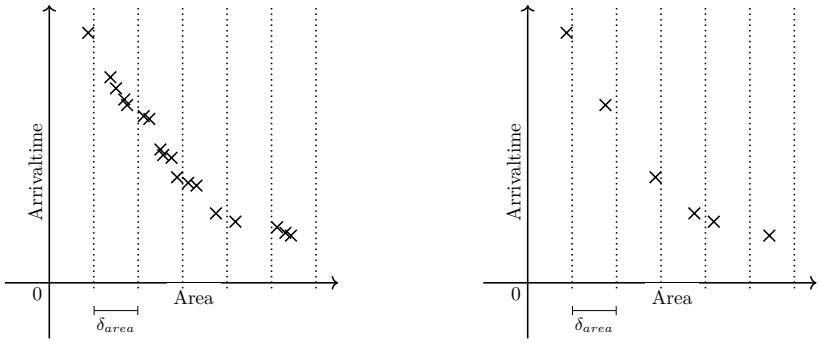

Abbildung 12: Einteilen und Filtern mit Buckets

Korollar 3.13. *Für Circuits C mit beschränkter Anzahl k an Highfanoutknoten sind $\mathcal{B}$ und $\mathcal{K}$ polynomiell beschränkt.*

Beweis. $\mathcal{B}$ ist beschränkt durch die Größe den größtmöglichen äquivalenten Circuit C_A von C geteilt durch δ_{area}. Es gilt $area(C_A) \leq area_{max} \cdot |V(C)|$, wobei $area_{max}$ die Größte des größten Elements der Library angibt. Daraus folgt $\mathcal{B} \in \mathcal{O}(area_{max}|V(C)|\frac{2(1-\lambda)}{\varepsilon})$.
Sei $v \in V(C)$ und K_v die Anzahl Klassen von v. Sei des Weiteren $K_{h,i}$ die maximale Anzahl Klassen eines Highfanoutknoten mit i weiteren Highfanoutknoten in seiner Cone. $\mathcal{K}_{\sqsubseteq}$ an einem Knoten v ist beschränkt durch die Anzahl aller möglicher Kombinationen von Kandidaten aller möglicher Klassen der Highfanoutknoten in $cone(v)$. Daraus folgt $\mathcal{K}_v \leq (\mathcal{B}\mathcal{K}_{h,k-1})^k$, da v maximal k Highfanoutknoten in seiner Cone besitzt, welche jeweils maximal $k-1$ Highfanoutknoten in Ihrer Cone besitzen. Des Weiteren gilt $\mathcal{K}_{h,k} \leq (\mathcal{B}\mathcal{K}_{h,k-1})^k$ mit der gerade genannten Begründung.
Es gilt $\mathcal{K}_{h,0} \leq 1$ da der Highfanoutknoten 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.

□

Beschränkt man die maximale Anzahl von Highfanoutknoten in C, so folgt aus dem obigen Korollar, dass die Anzahl der Klassen eines jeden Knoten und die Menge der sich darin befindlichen Kandidaten, polynomiell beschränkt ist.
Aus diesem Grund geht der im Folgenden vorgestellte polynomielle Algorithmus von einer beschränkten Anzahl von Highfanoutknoten aus.

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

3.6.2 Verknüpfen von Kandidaten

Ohne Highfanoutknoten lassen sich bei der Konstruktion eines Kandidaten für einen Knoten x, die Kandidaten k_i der Inputs i unabhängig voneinander wählen. Da das Klonen nicht erlaubt ist, ist dies bei Circuits mit erlaubtem höheren Fanout nicht möglich.
Die folgenden Bedingungen stellen sicher, dass Klonen verhindert wird.
Seien für einen Knoten v, O_v die Menge seiner offenen Knoten.

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 Inputkandidaten ein eindeutiger Kandidate festgelegt wird.
Bedingung 2 sichert diese Eigenschaft auch für die Inputs selber, denn es ist möglich, dass ein Inputknoten von x auch ein offener Knoten eines weiteren Inputs ist. Für diesen wird dadurch ebenfalls ein eindeutiger Kandidat festgelegt.
Somit sind alle Inputkandidatenmengen, welche diese beiden Bedingungen erfüllen, eine mögliche Grundlage für einen Kandidaten auf x.

3.6.3 Finden von Kandidaten

Für jedes der maximal $\mathcal{M}$ Matche eines Knoten v von einem Circuits mit den oben genannten Einschränkungen gilt folgende Vorgehensweise zur Findung aller passender 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 Kanidaten 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, entspricht aus dies polynomieller Laufzeit.

3.7 FPTAS

Erweitert man Circuits um die bisher in diesem Kapitel beschriebenen Eigenschaften, beschränkt durch k-Highfanoutgates, so gibt es für das folgende Problem einen FPTAS(fully polynomial time approximation scheme). Ein FPTAS ist ein Algorithmus, welcher, gegeben ein $\varepsilon > 0$, eine Lösung des Problems errechnet mit der Eigenschaft, dass für deren Kosten $c \leq (1+\varepsilon)OPT$ gilt. Hierbei sind OPT die Kosten einer optimalen Lösung des Technology

Mapping , bezüglich der bisher eingeführten Operationen.

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

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,$\varepsilon > 0$, k Highfanoutknoten mit beschränktem $fanout_{max}$, Tradeoff $\lambda \in [0,1]$

```
1  M ← finde_alle_matchings(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     filter_mit_buckets(g.tradeoff_curves, ε)
13 C' ← circuit(bester_kandidat(o, λ))
14 return C'
```

Hierbei wurde der Kern-Algorithmus um die in diesem Kapitel beschriebenen Routinen erweitert.
Für jeden Knoten werden vorbereitend alle, keine konstanten Kanten überdeckenden, Matche berechnet und seine offenen Knoten ermittelt. Daraufhin werden für alle Knoten, in topologischer Reihenfolge, und deren Klassen jegliche nicht dominierten Kandidaten hinzugefügt und die Tradeoffkurven daraufhin gefiltert. Es bleiben für jeden Knoten, aufgrund der Beschränkung auf k Highfanoutknoten, polynomiel 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 3.14. *FPTAS 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. Schritt 3 und 4 lässt 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. Highfanoutknoten, welche Inputs eines offenen Knoten sind, werden als offen deklariert und Highfanoutknoten, 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 5, 6, 7, 8, und 9 sind beschränkt durch $|V(C)|$, $\mathcal{M}$, $\mathcal{K}$, $\mathcal{B}$. Schritt 10 und 11 sind in $\mathcal{O}(\mathcal{K})$ implementierbar.
Die Menge der Kandidaten in den Tradeoffkurven sind beschränkt durch die Anzahl der Schleifendurchläufe der Schritte 7-11. Somit liegt die Laufzeit polynomiell in der der Schleifendurchläufe.
Schritt 13 ist durch die maximale Anzahl der Kandidaten einer Tradeoffkurve und $|V(C)|$ beschränkt.
Daraus folgt eine Gesamtlaufzeit von $\mathcal{O}(|V(C)|^2 + |V(C)|\mathcal{MBK}^2)$ □

3.8 Heuristik

Im folgenden Algorithmus ist die Menge der Knoten mit $fanout > 1$ beliebig groß. Die Anzahl der Highfanoutknoten bestimmt maßgeblich den Speicherbedarf an Kandidaten und die Laufzeit des FPTAS. Sie sind der Grund, warum sich der FPTAS für eine Anwendung des Technology Mapping auf einem gesamten Chip nicht eignet.
Von daher ist es ein naheliegender Ansatz für eine Heuristik, an jedem Highfanoutknoten nur einen Kandidaten zu speichern. Es ist jedoch wichtig aber nicht trivial, welchen man dort auswählt. Routinen für eine solche Auswahl werden in Kapitel 5 ausführlich behandelt. Im Folgenden wird beipielhaft eines dieser Verfahren erläutert.
Ist an jedem Highfanoutknoten ein Kandidat gewählt, so 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_matchings(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 Highfanoutknoten then
13         guess ← min_{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     filter_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 beschreiben ein einfaches Verfahren zur Auswahl eines Kandidaten für einen Highfanoutknoten. Angelehnt an den Kernalgorithmus wird nur der Kandidat behalten, welcher den Tradeoff minimiert. Dies garantiert keine optimale Lösung.
Die Menge an Kandidaten in einer Tradeoffkurve ist in der Praxis, durch die Einschränkung auf einen Kandidaten für Highfanoutknoten, deutlich geringer als im FPTAS. Aus diesem Grund lässt sich in der Praxis ohne Laufzeiteinbußen mit $\varepsilon = 0$ filtern.
Weitere Informationen und Erweiterungen dieser Heuristik befinden sich in Kapitel 5 und 9.
Schritt 19 entfehrnt die durch teilweise überfüssige Subcircuits evtl. hinzu-

gekommenen Buffer.

Laufzeit: Stellt man die gleichen Voraussetzungen an die Circuits wie der FPTAS und filtert mit einem $\varepsilon > 0$, so lässt sich diese Heuristik in der selben 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 DAGs mit mehreren Outputs

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

Da Signale von Outputknoten 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 Required Arrivaltimes

In Definition 2.6 wurde bereits der Begriff der Arrivaltime eines Knoten eingeführt. Dies ist die Zeit, zu welcher das letzte Signal bei einem Knoten ankommt. Diese Werte sind für die Inputknoten 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 Design Prozess eines Chips, gibt es neben der tatsächlichen Arrivaltime auch eine gewünschte Arrivaltime RAT (required AT), welche an den Outputs eines Graphen gegeben ist und ähnlich zur AT durch C propagiert wird. Somit ist sowohl AT und 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:

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

Die RAT der Outputs wird hierbei (wie das Delay der Inputknoten) als gegeben angenommen.

In der Praxis kommen die Signale oft später an als gewünscht. 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): Summe aller negativer Slacks der Outputs eines Circuits.

Letzterer ist in der Praxis gefragter, da eine sehr gute Verbesserung der SNS eine Verbesserung des WS in der Regel mit einschließt.

Angenommen man betrachtet einen Chip, dann 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 einem Input des Chips gelangt oder der Slack nicht mehr negativ ist.
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 füge hier mal ein Beispiel oder einen Verweis an zu einem äquivalenten Circuit C', mit geringerer Tiefe(Anzahl Kanten eines bzgl. Kantenenge längsten Weges in C), umformen. Diese umgebauten Instanzen besitzen nur einen Output, wenige Highfanoutknoten 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 in sehr schneller Laufzeit des Algorithmus. Der Nachteil jedoch ist, dass ein Chip oft sehr viele Wege besitzt, welche einen schlechten Slack realisieren und man somit den Chip nur inkrementell beschleunigt.

Eine weitere Herangehensweise für das Technology Mapping ist es, einen Circuit dahingehend zu optimieren, dass die SNS des Outputs minimiert wird. Dies ist jedoch bei den bisher betrachteten Circuits äquivalent zur Optimierung nach AT, da nur Instanzen mit einem Output betrachtet wurden und RAT für diesen eine Konstante ist.

4.2 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 besitzen. Folgende Umbauten sind notwendig, um mit dem Kern-Algorithmus auch diese Instanzen verbessern zu können.

Als erstes fällt auf, dass sich, wenn der Algorithmus für jeden Knoten die Kandidatenmenge errechnet hat, nicht einfach der beste Kandidat für den Output aus seiner Tradeoff-Kurve auswählen lässt. Dieser besitzt bei mehreren Outputs nämlich in der Regel offene Knoten. Jedoch ist bereits bekannt, wie man mehrere Kandidaten auswählt, so dass diese sich an den sich überschneidenden Knoten übereinstimmen. Somit lässt sich ein Circuit mit den bekannten Mitteln realisieren, welcher eine Kostenfunktion hinsichtlich Größe und WS optimiert.

Hieraus ergibt sich eine zweite Änderung. Bisher wurde das Delay eines Circuits C optimiert, indem das Signal des einen Outputs nach dem Umbau früher ankommt. Dies lässt sich auf einen Circuit mit mehreren Outputs übertragen. Da es mehrere Signale gibt, wählt man den Kandidaten des Outputs mit dem größten negativen Slack zuerst und die anderen folgen sortiert der Größe ihres Slacks nach (absteigend). Dies garantiert jedoch nicht, dass der WS des Circuits nach dem Umbau besser ist als vorher, da evtl. der Knoten der vorher den WS bildete, besser wird. Ein anderer Output könnte jedoch durch diesen Umbau schlechter werden.
Um dieses Problem zu umgehen, verändert man C vor dem Technology Mapping durch das Verbinden aller Outputs mit einem virtuellen Gate, mit nur einem möglichen Match (dem Gate an sich). Der veränderte Circuit lässt sich nun wie im Kern-Algorithmus optimieren und es wird automatisch das gerade dargestellte Problem gelöst.
Wie bereits in Kapitel 4.1 erläutert ist es in der Praxis profitabler die SNS des Circuits zu verbessern, anstatt den WS.
Also muss aus den Kandidatenmengen der Outputs derjenige Circuit-Kandidat gebaut werden, welcher die SNS minimiert.
Dieses Kriterium ersetzt, von diesem Punkt an, das 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.

5 Premapping von Highfanoutknoten

Der exponentielle Anstieg der Kandidatenmenge wird, wie in Kapitel 3.6 gezeigt, durch die Highfanout-Knoten verursacht. Daraus folgt, dass ein sehr hohes Laufzeit Potenzial in der Reduzierung der Kandidaten für diese Knoten liegt. Dieses Kapitel stellt mehrere Routinen für eine solche Reduzierung vor.

Die Kandidatenmenge eines jeden Highfanoutknotens wird, wie bereits in Kapitel 3.8 dargestellt, auf eins reduziert. Diese Routine wird auch das Premapping der Highfanoutknoten genannt. Hieraus folgt, dass jeder Knoten des Circuits C nur noch eine Klasse an Kandidaten besitzt, denn alle offenen Knoten seiner Cone sind Highfanoutknoten 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 Highfanoutknoten ein Kandidat zugewiesen wurde. Dadurch hat die Wahl der Premapping Routine eine zentrale Bedeutung in der Heuristik.
Im Folgenden werden drei verschiedene Routinen vorgestellt und auf deren Eigenschaften eingegangen. Genauere Informationen über die Unterschiede zwischen den Resultaten dieser Routinen sind in Kapitel 9 dargestellt.

5.1 Triviales Premapping

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

$$guess \leftarrow \min_{\text{Kandidat } k \text{ auf } v} \{\lambda AT(k) + (1-\lambda) area(k)\}$$

Diese Methode bietet sich an, lässt sich jedoch noch wie im Folgenden beschrieben weiter verbessern.

Im Gegensatz zu dem Kern Algorithmus aus Kapitel 2.2 kann auch mit $\lambda \in \{0,1\}$ keine optimale Lösung garantiert werden. Dies beweist Abbildung ??. Jedoch befindet sich die dadurch gefundene Lösung nahe an der optimalen. Nähere Informationen darüber, wie weit eine solche Lösung von der optimalen entfernt ist, siehe Kapitel 9.2.6.

5.2 Premapping durch Schätzen

Beim Premapping durch Schätzen wird versucht, eine Vermutung für die Kosten eines geeigneten Kandidaten aufzustellen. Ausgewählt wird dann der Kandidat, welcher die geringste Differenz zu den vermuteten Kosten besitzt. Die Schätzung erfolgt durch zwei Technology Mapping Läufe, welche einmal einen möglichst schnellen und einmal einen möglichst kleinen Circuit errechnen. Folgender Algorithmus wird für die Schrankenberechnung benutzt.

Algorithmus : 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) Matches auf v
 5 |   foreach Match m auf v do
 6 |   |   berechne schnellsten Kandidaten mit m auf v
 7 |   |   Update schnellster_(inv)_kandidat
 8 |   if v ist Highfanoutknoten 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 foreach Knoten v ∈ V(C') do
15 |   foreach Vorgänger p do
16 |   |   if p ist Highfanoutknoten then
17 |   |   |   inv(v,p) ← keine Invertierung
18 return C'
```

Dieser Algorithmus führt das Technology Mapping 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.
dieser algo stammt von Lukas (wie das tradeoff kurven bild und der folgende algo wie nehme ich das am besten mit rein ?

Der Untere Schranke Arrivaltime Agorithmus lässt sich in der gleichen Laufzeit wie die Heuristik aus Kapitel 3.8 implementieren, ist jedoch in der Pra-

xis, da nur ein Kandidat für jeden Knoten gespeichert wird, deutlich schneller.

Das Premapping durch Schätzen erfolgt nun durch folgenden Algorithmus:
algo nochmal genau ansehen und fragen klären

Algorithmus : Premapping durch Schätzen

```
1  C'_SNS ← untere_schranke_SNS(C)
2  C'_Area ← untere_schranke_Area(C)
3  SNS^- ← sum_of_negative_slacks(C'_SNS)
4  SNS^+ ← sum_of_negative_slacks(C'_Area)
5  α ← λ·SNS^- + (1−λ)·SNS^+ //Finale SNS Schätzung
6  foreach Highfanoutknoten v ∈ V(C) do
7      estim_small(v) ←neg_slack(v, C'_Area, α)
8      estim_fast(v) ←neg_slack(v, C'_SNS, α)
9      neg_slack_schätzung(v) ← λ·estim_fast(v) + (1−λ)·estim_small(v)
10
11 ...
12
13 if v ist Highfanoutknoten then
14     min ← min_{Kandidat k auf v} {|neg_slack(k) − neg_slack_schätzung(v)|}
15     foreach Kandidat k auf v do
16         if |neg_slack(k)-neg_slack_schätzung(v)| ≠ min then
17             lösche k
```

Die Schritte 1-9 werden an den Anfang der Heuristik für das TM mit Konvexkombination gesetzt. Die Schritte 13-17 ersetzen die Schritte 12-16 der Heuristik.
Es ist ausreichend alle möglichen Matche der Knoten nur einmal zu errechnen und diese Werte sowhl für die untere Schranke als auch für die Heuristik selber zu nutzen. Mit diesem Trick lassen sich beide Schranken in kurzer Zeit errechnen. Genauere Informationen zur Laufzeit und Güte der Premapping Routinen befindet sich in Kapitel 9.
Diese Routine liefert sehr gut Ergebnisse, da durch die Berechnung der Schranken das Potenzial eines Knoten errechnet wird, wie klein oder schnell Kandidaten dieses Knotens werden können. Mithilfe des Tradeoffparameter errechnet sich hieraus ein geschätzter Kostenwert. Es wird nur der Kandidat ausgewählt, welcher die Differenz zu dieser Schätzung minimiert.

5.3 Erweitertes Premapping durch Schätzen

gilt es noch zu entwickeln

6 Preprocessing

Die Möglichkeiten durch das Matching sind im Allgemeinen 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 die Gates nach folgender Routine zerlegt. hier das huffman coding und ein Beispiel mit einbringen und decomposing begriff einführen
Der Vorteil des Decompose ist, dass die Möglichkeiten des Technology Mapping 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 Decomposen finden sich in Kapitel 9.

7 Weitere Optimierungskriterien

Das Technology Mapping arbeitet im Chip-Design auf realen Instanzen. Dadurch kommen zu den bereits vorgestellten weitere mögliche Optimierungskriterien hinzu. Es handelt sich hierbei einmal um Kriterien, welche in die Kostenfunktion mit eingebracht werden können und somit während des Technology Mapping 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. Weitere Kriterien folgen.
Des Weiteren sind viele Gates, zumindest 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 Geschwindigleitsvorteile geschaffen werden können.

7.1 pinabhängiges Delay

Bis hierhin war das Delay eines Gates als eine nicht negative reelle Zahl definiert. Die meisten Gates besitzen mehr als 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, so 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 Dealy des Outputs verbessern. Aus diesem Grund ändert sich die AT eines Knotens wie folgt:

Definition 7.1. AT mit pinabhängigen Delay:
Sei C ein Circuit und $v \in V(C)$.
Die AT von v mit pinabhängigen 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. Um die Optimilität des, noch vorzustellenden, allgemeinen Algorithmus zu wahren, wird ein Kandidat für jede mögliche Permutation der Inputs gespeichert, falls dieser nicht dominiert ist.
Nach aktuellen 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.

7.2 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, besitzt 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 natürlich 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.
In Kapitel 9 finden sich weitere Ausführungen über die Auswirkungen von diesem Austausch in der Kostenfunktion.

7.3 Layer Assignment

Die Knoten eines Circuits sind durch Kanten miteinander verbunden. Die ausgehenden Kanten eines Knotens bilden ein Netz und die Endknoten der Kanten dessen Menge von Terminalen. Diese werden auf dem Chip zu einem Steinerbaum verbunden. Des Weiteren ist ein Chip in mehrere Schichten (Layers) unterteilt, in welche die Kanten physikalisch eingebettet werden. Das Verlegen einer Kante in einem Layer bringt Kosten mit sich, welche abhängig von der Wahl des Layers sind. Jedem Netz ist nun eine Menge von Schichten zugeordnet, in welche die Kanten des Steinerbaumes gelegt werden dürfen.
Durch die vorgestellten Technology Mapping Algorithmen werden in dem umgebauten Circuit C' Netze von Knoten, über die gematcht wurde, nicht mehr benötigt. Für Gates, über deren Nachfolger gematcht wurde, verändert sich jedoch die Terminalmenge des dazugehörigen Netzes. Den Netzen aus C' muss nun wieder eine Menge von Layern zugeordnet werden, so dass die Terminalmengen untereinander auf dem Chip verbunden werden können und die zusätzlichen Kosten nicht übermäßig groß werden.
Wenn sich ein Terminal eines Netzes N mit Layermenge L ändert, wurde über den darüberliegenden Knoten gematcht und dessen Netz N' mit Layermenge L' ist verschwunden. Um sicherzustellen, dass in N jedes Terminal über die Layer erreichbar ist, wird aktuell $L := L \cup L'$ gesetzt. Dies geschieht für jedes Netz M aus C' und dessen geänderte Terminale und garantiert die Existenz eines Steinerbaums in M.
Es muss noch überprüft werden, ob durch eine geänderte Zuweisung der Layermenge der Netze von C' geringere Kosten bei der Realisierung der Netze garantiert werden können. Da dies bisher die beste Methode ist das Problem an dieser Stelle zu lösen, und in den getesteten Instanzen die

zusätzlichen Kosten durch die Zuteilung der Kanten nicht übermäßig hoch sind, werden weder diese Kosten noch das Layer Assignment in dieser Arbeit weiter behandelt.

8 Ressource Sharing

In diesem letzten theoretischen ? Kapitel wird ein anderer Ansatz für eine Heuristik vorgestellt. Da dieser Ansatz noch nicht implementiert wurde, wird in Kapitel 9 nicht darauf eingegangen.
Es folgt die allgemeine Definition des Problems und daraufhin eine Heuristik, welche sich dieses Problem zunutze macht.

Es handelt um das Ressource Sharing Problem. Eine Instanz des Problems besteht aus einer endlichen Menge von Kunden $\mathcal{C}$, von denen jeder eine Aufgabe erledigen möchte. Jeder Kunde $c \in \mathcal{C}$ besitzt ein Spektrum an Vorgehensweisen $\mathcal{B}_c$ um seine Aufgabe zu meistern. Hierbei ist $\mathcal{B}_c$ (Block genannt) eine konvexe Menge. Jede Vorgehensweise benötigt Ressourcen für ihre Umsetzung.
Sei $\mathcal{R}$ die endliche Menge aller verschiedener Ressourcen. Des Weiteren sei g die Funktion, welche für jeden Kunden c und $b_c \in \mathcal{B}_c$, die benötigte Menge einer jeden Ressource $r \in \mathcal{R}$ angibt. Es gilt also $\forall c \in \mathcal{C} : g_c : \mathcal{B}_c \to \mathbb{R}^{\mathcal{R}+}$.
Ziel des Ressource Sharing ist es nun, jedem Kunden eine Vorgehensweise zuzuordnen, mit welcher er seine Aufgabe erledigt. Dabei wird über den Verbrauch der am meisten genutzten Ressource minimiert.
Es folgt die formale Definition des Problems.

RESSOURCE SHARING PROBLEM

Instanz: Endliche Mengen $\mathcal{R}$ von Ressourcen und $\mathcal{C}$ von Kunden. Einen, durch eine endliche Menge repräsentierten, konvexen Block $\mathcal{B}_c \forall c \in \mathcal{C}$ und eine konvexe Funktion $g_c : \mathcal{B}_c \to \mathbb{R}_+^{\mathcal{R}} \forall c \in \mathcal{C}$.

Aufgabe: Finde $\forall c \in \mathcal{C} b_{\in} \mathcal{B}_c$, welche λ^* so nah wie möglich kommen. Dabei gilt:

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

hier vielleicht noch ein einfaches beispiel (evtl) und auf jeden fall der Verweis zum paper und ein Vermerk auf die Benutzung im Chipdesign
daraufhin der Technology Mapping ansatz! theoretische Beweise?

9 Qualitäts- und Laufzeit-Analyse

In diesem Kapitel werden die vorgestellten Algorithmen bezüglich Laufzeit und Güte analysiert. Vorher ist es jedoch Notwendig ein paar Angaben zu den bearbeiteten Instanzen, um die Ergebnisse der Algorithmen besser einordnen zu können.

9.1 Struktur realer Instanzen

Die Instanzen eines Chips sind im Folgenden 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 besteht aus sehr vielen Routinen, welche aus einem Bauplan einen produzierbaren Chip designen. Auf diesem Weg gibt es viele Zwischenstände (Snapshots genannt). 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 Highfanouknoten, welche keine Inputs sind, eines Circuits C und NI_C die Menge aller Knoten abzüglich der Inputknoten von C. Der durchschnittliche Anteil von H_C an NI_C beträgt $30,95\%$. Die Menge der Highfanouknoten, welche ebenfalls Inputs sind wurde hier nicht beachtet, da 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 werden ebenfalls auf einen Kandidaten beschränkt oben erwähnt ?, da Sie zu den offenen Knoten darauffolgender Knoten gehören. Da diese nur einen solch geringen Anteil an allen Outputs be-

$\lvert V(C)\rvert$	**#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 13: Größverteilung der getesteten Instanzen

sitzen werden nur wenige Outputs so
früh festgelegt. Auch hier wurden Outputs, welche auch Inputs sind, aus den gleich Gründen wie oben, nicht betrachtet.
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 13 zeigt die Verteilung der getesteten Instanzen hinsichtlich der Größe ihrer Knotenmenge.
Kandidaten menge kann hier noch hin also menge der Kandidaten abh von Knotenmenge bzw anzahl highfanoutknoten und abh von den oben genannten zusatzfeatures (zb pinabh. delay, Library)
und dass vielleicht die varianz (anhand eines bildes beweisen?) des hoghfanout anteil ziemlich klein ist und somit Knotenmenge und highfanoutmenge ansich ausreichend aussagekräftig sind

9.2 Analyse der Ergebnisse

Es folgt eine Anlayse der Ergebnisse des Algorithmus hinsichtlich der verschiedenen, oben vorgestellten Variationen.

9.2.1 Tradeoffparameter

Area, SNS, WS oder Energieverbrauch eines Circuits liegen nicht unbedingt in derselben Größenordnung. ¡- rauslassen?Im Folgenden wird ausgehend von der Technology Mapping Heuristik ohne Präpozessing, und mit dem Premapping durch Schätzen der Einfluss des Tradeoffparameters λ, abhängig von einer Optimierung nach kommt das noch? -¿Power oder Area, auf die Änderung der Kosten untersucht. Die Kostenfunktion ist hierbei $\lambda SNS(C) + (1-\lambda)Area(C)$ bzw. $\lambda SNS(C) + (1-\lambda)Power(C)$
Abbildung 14 veranschaulicht die durchschnittliche prozentuale Verbesserung der Kosten abhängig von λ.

Hier ist nur eine schwache abhängigkeit bezüglich des Tradeoffparameters zu beobachten. 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 Kanidat mit nahezu dengleichen Kosten geschätzt wird wie ein kleinerer und somit unabhängig des λ sehr kostenähnliche Kandida-

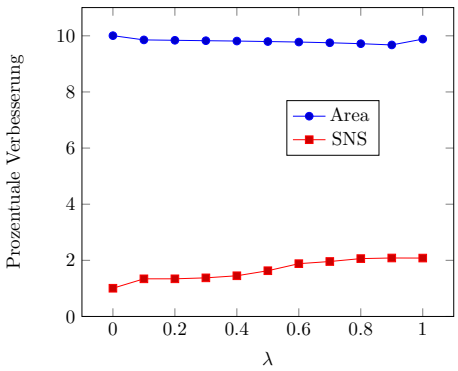

Abbildung 14: Einfluss von λ

ten gewählt werden, wodurch die Kosten des äquicalenten Circuits sich in abhängigkeit des λ nur gering verändern.
Unabhängig von λ zeigt die Abbildung jedoch auch ein enormes Verbesserungspotenzial des Circuits in SNS und $Area$. In folgenden Tests wird $\lambda = ?$ gesetzt, da dies in den getesteten Instanzen die größte Verbessung im Schnitt gebracht hat.

Offenbar bringen 0,lala 0,trlala die besten Verbesserungen. Im Folgenen sei λ durch diese Werte vorgegeben.
was ist mit lambda abh. von diesem verhätnis wählen ?

9.2.2 Premapping

9.2.3 Preprozessing

inclusive der graphen größe vergleiche

9.2.4 Power und Area Vergleich

9.2.5 Bucket filetering ε im vergleich

9.2.6 Gütevergleich kleiner öptimal ”gel instanzen

9.2.7 Verhalten weiterer Kosten

die anfallenden kosten die nicht im Technology Mapping gemessen werden beobachten und in der future work dran anschließen

9.2.8 Zusammenfassung?

oder am ende nur ein laufzeit güte tradeoff mit eigenem Unterkapitel ?

9.3 Laufzeitanalyse

Aus der theoretischen Laufzeitschranke aus Kapitel 3.8 folgt, dass die Laufzeit maßgeblich von der Menge der Highfanoutknoten abhängt. Andere Faktoren wie das Aufteilen von Gates, die Größe der Library oder die Wahl der Premapping subroutine spielen ebenfalls eine wichtige Rolle. Dies verdeutlicht die Übersicht ??.
Bezüglich der Größe der Library lassen sich die Chips, mit Ausnahme weniger Unterschiede in zwei Gruppen einteilen. Die markierte Teilmenge der Abbildung ?? entspricht der Menge der (so genannten) komplexen Gates. Ein Chip lässt sich in der Praxis, abhängig davon, ob er die komplexen Gates grundsätzlich erlaubt, einer der zwei Gruppen zuordnen. Diese Unterscheidung findet sich ebenfalls in Abbildung ??.

9.3.1 Globale Laufzeitanalyse

welche größenordnungen sind überhaupt lösbar

9.3.2 Lokale Laufzeitanalyse

wie unterscheiden sich die einzelnen Varianten in der Laufzeit?
oder lässt sich beides gut in einem bild erkennen ?

hier auch die decompose laufzeit vgl mit rein ?

9.3.3 Bucket filetering ε im vergleich

9.3.4 Laufzeitverlgeich kleiner öptimal "gel instanzen

9.4 Güte laufzeit vergleich

9.4.1 Kleine Instanzen

9.4.2 Allgemein

wahrscheinlich ist der unterschied bei decompose und ohne am größten den mit der besten premapping methode durchführen

9.4.3 Güte Bucket filetering ε

10 Fazit und Ausblick

11 Literaturverzeichnis

[Elb17] Lucas Elbert. Approximationsalgorithmen für das Technology-Mapping, 2017.

[HNKF18] David A. Hounshell Hassan N. Khan and Erica R. H. Fuchs. Science and research policy at the end of moore's law. *Nature Electronics*, 1, January 2018.

[Keu87] K. Keutzer. Dagon: Technology binding and local optimization by dag matching. In *24th ACM/IEEE Design Automation Conference*, pages 341–347, June 1987.

[KR89] Keutzer and Richards. Computational complexity of logic synthesis and optimization. In *International Workshop on Logic Synthesis*, 1989.

[Moo65] Gordon Earle Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38, April 1965.

[Pos41] Emil Leon Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematics*, 5, 1941.

[Tra15] Khai Van Tran. Algorithmen für das Technology-Mapping, 2015.