

# Heuristiken für das TechnologyMapping

Alexander Zorn

Geboren am 26. Mai 1996 in Bonn

21. Juni 2018

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Stephan Held

Zweitgutachter: YYYY YYYY

FORSCHUNGSINSTITUT FÜR DISKRETE MATHEMATIK

MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT DER  
RHEINISCHEN FRIEDRICH-WILHELMS-UNIVERSITÄT BONN



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
<b>2</b>	<b>Terminologie &amp; grundlegender Algorithmus</b>	<b>4</b>
2.1	grundlegende Definitionen . . . . .	4
2.2	Kern Algorithmus . . . . .	6
<b>3</b>	<b>FPTAS und Heuristik</b>	<b>9</b>
3.1	Tradeoffprobleme . . . . .	9
3.2	Highfanoutknoten . . . . .	10
3.2.1	Klonen . . . . .	13
3.2.2	Verknüpfen von Kandidaten . . . . .	13
3.3	zu lange Kanten . . . . .	14
3.4	Teilweise überflüssige Subcircuits . . . . .	14
3.5	Matchingprobleme . . . . .	16
3.5.1	Anzahl Matchings . . . . .	16
3.5.2	Matching Suche in polynomieller Zeit . . . . .	17
3.6	FPTAS und Heuristik . . . . .	17
3.6.1	Filtern mit Buckets . . . . .	17
3.6.2	FPTAS . . . . .	18
3.6.3	Heuristik . . . . .	19
<b>4</b>	<b>Mehrere Outputs</b>	<b>20</b>
4.1	required Arrivaltimes . . . . .	20
4.2	Mehrere Outputs . . . . .	22
<b>5</b>	<b>Premapping von Highfanoutknoten</b>	<b>23</b>
5.1	triviales Premapping . . . . .	23
5.2	erweitertes Premapping . . . . .	24
5.3	Premapping durch Schätzen . . . . .	24
<b>6</b>	<b>Präprozessing</b>	<b>24</b>
<b>7</b>	<b>Weitere Optimierungskriterien</b>	<b>24</b>
7.1	pinabhängiges Delay . . . . .	25
7.2	Power Optimierung . . . . .	25
7.3	Layer Assignment . . . . .	26
<b>8</b>	<b>Version der Heuristik, welche obige Kriterien beherzigt</b>	<b>27</b>
<b>9</b>	<b>Ressource Sharing</b>	<b>27</b>

<b>10 Laufzeitanalyse</b>	<b>28</b>
10.1 Struktur realer Instanzen . . . . .	28
10.2 Analyse der Ergebnisse . . . . .	28
10.3 Laufzeitanalyse . . . . .	29
<b>11 Fazit und Ausblick</b>	<b>29</b>

# 1 Einleitung

Der zunehmende Gebrauch elektronischer Geräte verlangt nach immer leistungsfähigeren Computerchips. Ein solcher wenige Quadratzentimeter große Chip beherbergt bis zu mehreren Milliarden Transistoren, welche, durch Drähte verbunden, gemeinsam eine Logische Funktion errechnen. Das Chipdesign beschreibt die Aufgabe aus einer gegebenen Logischen Funktion einen herstellbaren Chip zu entwerfen, welcher diese 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 (links) zeigt dies an einem kleinen Beispiel. Die Realisierung einer solchen Funktion ist jedoch nicht eindeutig, wie die in Abbildung 1 (links und rechts) gezeigte Nachbildung, beweist.

Die Größe der Menge aller möglicher Baupläne (später Circuit) für eine Logische Funktion hängt maßgeblich

von der Anzahl der zur Verfügung stehenden Bauteile, sowie von dem Aufbau der Funktion, ab. Es stellt sich heraus, dass im Allgemeinen eine Vielzahl möglicher Realisierungen einer Logischen Funktion existieren. Jedes Bauteil besitzt physikalische Eigenschaften an Größe, Geschwindigkeit (Delay) etc.. Somit besitzt auch jede Realisation solche Eigenschaften.

Ziel des TechnologyMapping ist es für eine Logische Funktion eine Realisierung zu finden, welche eine Kostenfunktion (bestehend aus den physikalischen Eigenschaften) optimiert. Die Wahl der Implementierung hat direkte Auswirkungen auf die Schnelligkeit, Größe und den Stromverbrauch des fertigen Chips. Hierbei geht das TechnologyMapping von einer bereits realisierten Logischen Funktion aus und baut diese um zu einer möglichst kostengünstigen Alternative um.

Der optimale mögliche Umbau lässt sich bei kleinen oder eingeschränkten gegebenen Bauplänen noch in akzeptabler Zeit finden. Die Lösung dieses Problem 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 akzeptabler Zeit einen möglichst kostengünstigen Umbau ermöglicht.

am ende hier noch eine kurze Quellenübersicht geben an lukas orientiert

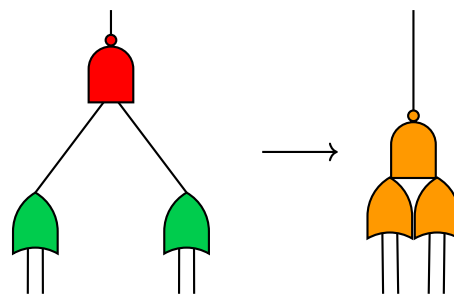


Abbildung 1: Zwei Realisierungen der Logischen Funktion  $\neg((w \vee x) \wedge (y \vee z))$

## 2 Terminologie & grundlegender Algorithmus

### 2.1 grundlegende Definitionen

Es folgen ein paar grundlegende Definitionen zur 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  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  mit  $n$  Inputs und  $m$  Outputs.

**Definition 2.2.** Gate und Library:

Ein Gate  $g$  mit Eingangsgrad  $n \in \mathbb{N}$  ist ein Tripel  $(f_g, d_g, area_g)$ . Hierbei sind  $d_g, area_g \in \mathbb{R}_{\geq 0}$ . Des Weiteren gilt  $f_g$  ist eine boolesche Funktion mit  $f_g : \{0, 1\}^n \rightarrow \{0, 1\}$ .

Eine Library  $L$  ist eine Menge von Gates und sei  $f_{anin_{max}} := \max\{arity(g) | g \in L\}$ .

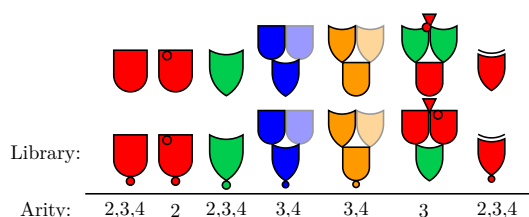


Abbildung 2: Beispiel einer Library

$area_g$  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 kann. Hierauf wird jedoch erst in Kapitel 3 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 folgenden Eigenschaften. Jeder Knoten gehört zu einer der aufgelisteten Kategorien:

- **Input** Knoten mit Eingangsgrad Null.
- **Gates** mit mindestens einer eingehenden Kante und ausgehenden Kante. Diese korrespondieren zu der Definition oben mit dem Zusatz dass an jedem der Inputs optional ein Inverter liegen kann.

- **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 originalen ist. Dies formalisieren die folgenden Definitionen.

**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 der Circuit und eine Bijektion  $f : X \rightarrow inputs(p)$
- Ein Funktion  $inv : inputs(p) \rightarrow \{not\_inv, inv\}$

So dass der Circuit  $C'$ , welcher durch den Austausch des Sub-Circuits 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, sodass  $\forall o \in O \exists! h \in K : h$  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.

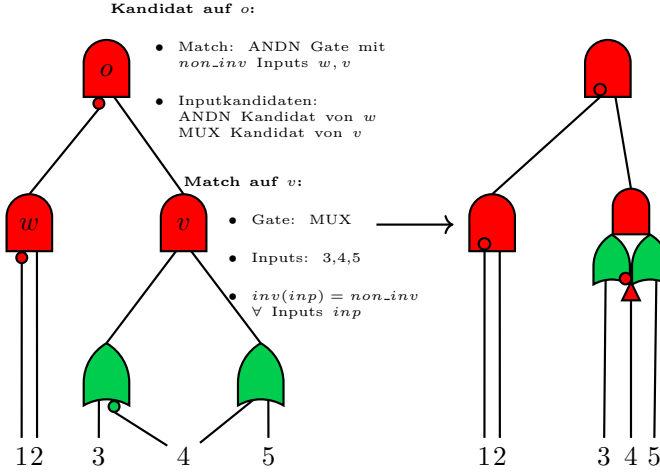


Abbildung 3: Beispiel einer Library

Ein Circuit-Kandidat  $C$  ist eine Möglichkeit den Circuit physikalisch zu realisieren. Wie bereits in der Einleitung bemerkt gilt es nun den besten Kandidaten auf  $C$  auszuwählen. Dafür ist ein Maß für Implementierungen von Circuits notwendig. Es folgen zwei geläufige Beispiele. In der Praxis (und im späteren Verlauf dieser Arbeit) wird in der Regel eine convex-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:

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

Wobei  $can(K)$  die Menge der Kandidaten von  $K$  sind,  $area_{inv}$  die Größe eines Inverters ist und  $inputs(k)$  sind die Inputknoten des Outputknoten des Kandidaten  $k$ . 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 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 Arrivaltme/Area minimiert.

**Algorithmus :** (einfaches) Technology Mapping

**Input :** Circuit  $C$  kreisfrei mit finalem Output  $o$ , Library  $L$

```
1 bester_kandidat[]  $\leftarrow \emptyset$ 
2 bester_inv_kandidat[]  $\leftarrow \emptyset$ 
3 foreach Knoten  $v \in V(G)$  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[ $o$ ]
```

**Definition 2.7.** optimaler TechnologyMapping Algorithmus:

Ein Algorithmus für das TechnologyMapping auf einem Circuit  $C$  heißt optimal, wenn er den (bzgl. der Kostenfunktion) besten äquivalenten Circuit  $C'$  liefert, welcher durch das Anwenden der erlaubten Operationen auf  $C$  konstruiert werden kann.

Daraus folgt, dass der Zusatz optimal abhängig davon ist, was die erlaubten Operationen sind. Im Weiteren Verlauf der Arbeit werden weitere Operationen hinzugefügt und der Circuit verallgemeinert. Das Attribut optimal bezieht dann, wenn dies nicht dabei steht, auf alle bisher vorgestellten Operationen. Zu diesem Punkt umfasst die Menge der erlaubten Operationen das logische äquivalente matchen von Subcircuits mit Elementen der Library.

Ein optimaler TechnologyMapping 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 TechnologyMapping

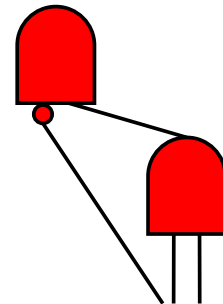


Abbildung 4: Ein Circuit dessen boolesche Funktion  $f = 0$  ist

nicht möglich.

**Korollar 2.8.** *Das einfache TechnologyMapping ist optimal.*

*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 Highfanout-Knoten überschneiden sich diese nicht. Für jeden Knoten und jedes Match gibt es nur einen Kandidaten zur Auswahl, da für die Inputs des Matche 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 für alle Knoten mit kleinerem topologischen Rang als  $rand(v)$  ist der bestmögliche Kandidat gespeichert, so wird, nach gerade beschriebenem Vorgehen, 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 Candidate, 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.  $\square$

**Korollar 2.9.** *Der Algorithmus für das (einfache) TechnologyMapping besitzt  $\mathcal{O}(|V(C)||L|)$ -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  $\mathcal{O}(fanin_{max}!)$  errechnen. Da  $fanin_{max}$  als beschränkt gegeben ist, entspricht dies  $\mathcal{O}(1)$ . **Beweis ?**. Schritt 6 ist, wie bereits erwähnt, schnell implementierbar, da für jeden der  $\max fanin_{max}$  Inputs nur der beste Kandidat 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  $\mathcal{O}(|V(C)||L|)$ .  $\square$

### 3 FPTAS und Heuristik

#### 3.1 Tradeoffprobleme

Der oben vorgestellte Algorithmus ist in der Lage den bestmöglichen Umbau eines eingeschränkten Circuits zu bezüglich Area oder Delay zu errechnen. Es existiert ein Tradeoff zwischen Area und Delay. Dies hat zur Folge, dass ein möglichst kleiner Circuit im Allgemeinen sehr langsam ist und man bei einer sehr schnellen Lösung mit einem großen Platzverbrauch rechnen muss. In der Anwendung des TechnologyMapping ist jedoch weder ein sehr langsamer noch ein besonders grosser Circuit akzeptabel.

Daraus folgt die Nachfrage nach einem Algorithmus, welcher in der Lage ist bezüglich einer Konvexkombination oder einer Schranke zu optimieren. Daraus ergeben sich die beiden folgenden Optimierungs-Probleme:

##### TECHNOLOGYMAPPING MIT KONVEXKOMBINATION

**Instanz:** Circuit  $C$ , mit einem Output, Library  $L$  mit beschränktem  $fanin_{max}$ ,  $|L|$  beschränkt und Tradeoff-Parameter  $\lambda \in [0, 1]$

**Aufgabe:** Finde einen Circuit-Kandidaten  $K$  auf  $C$ , welcher  $\lambda AT(K) + (1 - \lambda) area(K)$  minimiert.

##### TECHNOLOGYMAPPING 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.

Im weiteren Verlauf dieses Kapitels, werden diese Problemstellungen auf Circuits mit mehreren Outputs erweitert.

Diese **diese Probleme sind äquivalent Beweis? oder verweis aus quelle erwähnen dass da die äquivalent nur noch Konvexkombination**

Dadurch ergibt sich folgende Problemstellung für den Algorithmus: Angenommen an jedem Knoten  $v$  würde, wie im Kern-Algorithmus, nur derjenige Kandidat gespeichert werden, welcher die Kostenfunktion an  $v$  optimiert. Dadurch 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. Dazu 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 TechnologyMapping verwendet offenbar (in einem Korollar beweisen ?) nur nicht-dominierte Kandidaten, woraus 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.

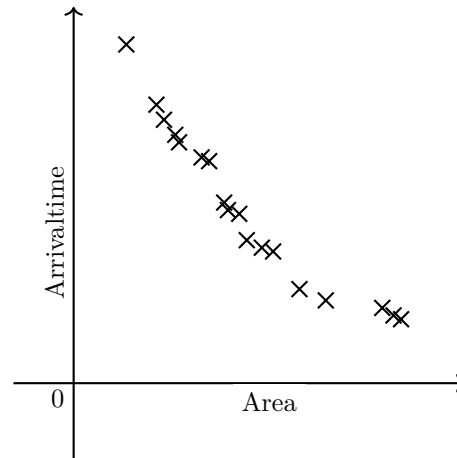


Abbildung 5: Ordnen der Kandidaten in einer Tradeoffkurve

Die beiden vorgestellten Probleme sind NP-vollständig. Daraus folgt, dass sich ab diesem Punkt wahrscheinlich kein polynomieller optimaler Algorithmus für das TechnologyMapping finden lässt. Dadurch dass sich zwei Kandidaten in den meisten Fällen nicht mehr vergleichen lassen, wird eine Vielzahl von Kandidaten an jedem Knoten gespeichert. Dies zeigt sich in einem exponentiell großen Speicheraufwand.

Ein Beweis der NP-vollständigkeit findet sich in [hier den verweis zu einem Beweis einfügen](#).

### 3.2 Highfanoutknoten

Der oben beschriebene Kern Algorithmus arbeitet nur auf Circuits, in denen keine Highfanoutknoten existieren. Diese Eigenschaft kommt auf einem realen Chip jedoch sehr häufig vor (ca. 25% der gesamten Knoten sind Highfanoutknoten für einen genaueren Zusammenhang von der Anzahl der Highfanoutknoten und der Laufzeit siehe das Kapitel 10).

Es ist möglich einen Circuit, in kleinere Subcircuits zu unterteilen, welche solche Highfanoutknoten nicht besitzen. Die Subcircuits werden einzeln mit dem Algorithmus (sehr schnell) optimiert und daraufhin zu einem C äquivalenten Circuit C' zusammengesetzt. Diese Vorgehensweise findet sich ausführlich in [Hier das eine Pa-per einsetzen](#) wieder. Abbildung 6 ver-bildlicht diesen Ansatz einer Heuristik. 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 auf diese Art der Heuristik nicht mehr eingehen.[zu anna gibt es sonst noch einen grund dies nicht doch einmal auszuprobieren?](#)

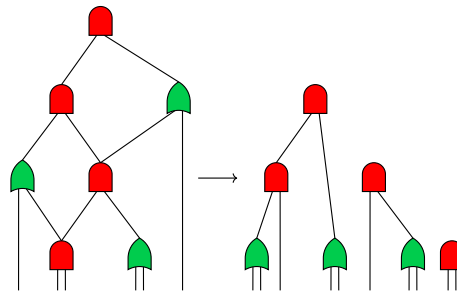


Abbildung 6: Unterteilen eines Circuit in Highfanoutfreie Subcircuits

Das Kern-Problem der Highfanoutknoten ist, 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 Knoten  $w$  die Wahl der Kandidaten der Input-Kandidaten von  $w$  nicht unabhängig von einander sein muss.

Abbildung 7 zeigt zudem ein weiteres Problem der Implementierung auf. Die Anzahl der zu Speichernden Kandidaten kann, mit beliebig vielen

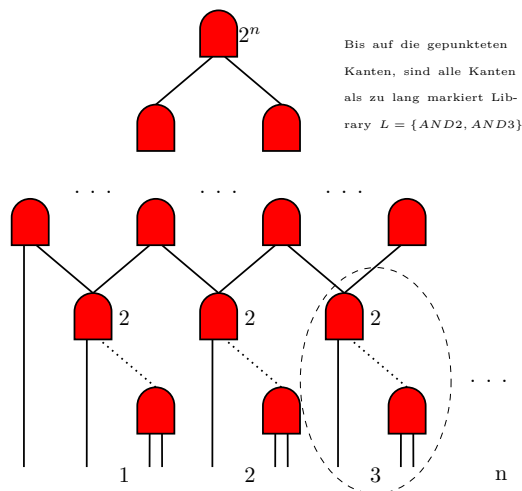


Abbildung 7: Exponentiell viele Kandidaten bereits bei sehr eingeschränkter Library

Highfanoutknoten in  $C$ , exponentiell bezüglich  $|V(C)|$  sein. Daraus folgt ein Implementierungsproblem, auf welches im Weiteren Verlauf dieser Arbeit noch eingegangen wird.

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  $v$ :

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

**Definition 3.3.**

Sei  $C$  ein Circuit und  $v \in V(C)$ . Dann wird die durch  $cone(v)$  berechnete Funktion. Die **bis v berechnete Funktion** genannt.

**Definition 3.4.** Offene Knoten:

Sei  $C$  ein Circuit und  $v, w \in V(C)$ . Dann heißt  $w$  offener Knoten von  $v$ , wenn folgendes gilt:

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

Mit anderen Worten 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. Von dieser Menge ist  $v$  selber ausgenommen. Dies sind gerade die Highfanout-Knoten, 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 offene Knoten gehören muss), bereits eindeutig definiert.

Abbildung 8 visualisiert die vorangegangenen Definitionen.

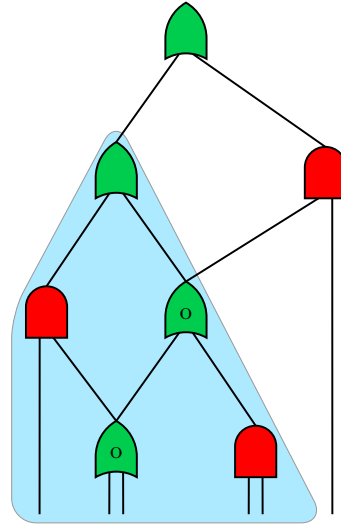


Abbildung 8: Visualisierung der Definitionen 3.2 bis 3.4

**Definition 3.5.** 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.

Nach der einführenden Erläuterung 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 somit mit Highfanoutknoten arbeiten zu können, werden für jeden Knoten  $v$  und jede Klasse von  $v$  in dem optimalen Algorithmus, alle nicht dominierten Kandidaten gespeichert. Daraufhin ist der noch verbleibende beste Kandidat des Outputs die beste Lösung.

Dabei wird, zur Speicherung der Kandidaten, für jede Klasse eines Knotens eine Tradeoff-Kurve angelegt.

### 3.2.1 Klonen

Oben wurde erwähnt, dass Kandidaten zweier Knoten an den offenen Knoten übereinstimmen müssen. Tun Sie das nicht, so werden offenen 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 gilt. Der erhöhte Verbrauch von Kanten bringt höhere Routing Kosten mit sich, welche im TechnologyMapping nicht beachtet 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.

### 3.2.2 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 die an den offenen Knoten der Inputkandidaten ein eindeutiger Kandidat 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  $x$ .

### 3.3 zu lange Kanten

Abbildung 9 veranschaulicht eine häufig auftretende Situation. Es handelt sich 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.

Weiter unten wird eine zusätzliche Klasse von Kanten eingeführt über welche man nicht matchen darf. Diese Kanten bezeichnet man als konstant. Um diese Situation zu vermeiden, wird bei der Bildung jedes Matches darauf geachtet über keine konstante Kante zu matchen.

Durch die Hinzunahme der zu langen Kanten zu den konstanten Kanten, kann keine optimale Lösung mehr im allgemeinen Algorithmus garantiert werden, von daher wird dies im optimalen Algorithmus nicht gemacht, bei der darauffolgenden Heuristik jedoch schon.

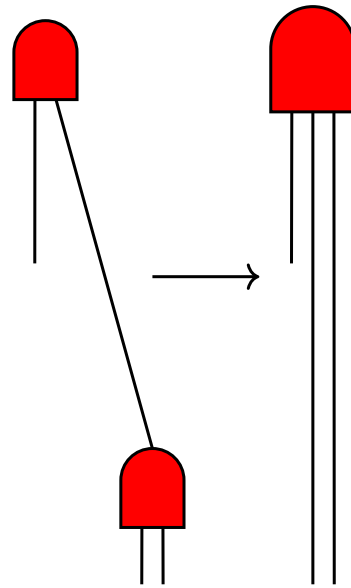


Abbildung 9: Visualisierung eines Matches über eine lange Kante

### 3.4 Teilweise überflüssige Subcircuits

In der Abbildung 4 lässt sich erahnen, dass nicht unbedingt alle Inputs eines Circuits relevant sind für die Outputs. Zur genaueren Einordnung folgt eine Definition.

**Definition 3.6.** vollständig überflüssiger und teilweise überflüssiger Circuit  
Sei  $C$  ein Circuit mit Logischer Funktion  $f : \{0,1\}^n \rightarrow \{0,1\}^m$ .  $C$  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 bedeutet, 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. Dies lässt sich jedoch weiter verbessern, 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 der aktuellen Architektur

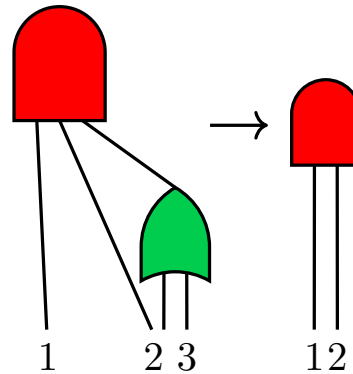


Abbildung 10: Nur Input 1 und 2 sind die relevanten Inputs

des TechnologyMapping Algorithmus. In der Praxis ist das Vorkommen von vollständig überflüssige Circuits verschwindend gering. Von daher werden die vollständig überflüssigen Subcircuits nicht mehr behandelt und kommen auch in den Folgenden Algorithmen nicht vor und zählen auch nicht zu den Kriterien des optimalen TechnologyMapping .

Im Gegensatz dazu kommen die teilweise überflüssigen Circuits sehr wohl vor. Bei der Konstruktion eines Chips passiert 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 TechnologyMapping vorhanden und kann vermieden werden. Von daher wird in diesem Fall kein Buffer, sondern nur die Kanten vom Input des Buffers zu seinen Outputs gebaut. Dies verhindert den Einbau eines nicht nötigen Gates. Diese zusätzliche Bearbeitung läuft von nun an in jedem folgenden Algorithmus automatisch im Hintergrund und findet keine Erwähnung mehr.

Obwohl nun das Gegenbeispiel aus Kapitel 2.2 nicht mehr gültig ist, garantiert auch hier ein optimaler Algorithmus keine bestmögliche Implementierung der eines Circuits zugrunde liegenden logischen Funktion.

### 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 einen Algorithmus der auf realen Instanzen fast alle möglichen Matche findet und in polynomieller Zeit implementierbar ist.

#### 3.5.1 Anzahl Matchings

hier noch auf professor helds idee eingehen!! mit dem suchen nach matchen durch durchgehen aller möglichkeiten von schnitten und die dissertation verlinken sowie eine Schranken für die anzahl der Matche angeben Folgendes Lemma legt eine äquivalente Definition für Matche nahe. Sei, gegeben ein Circuit  $C$  ohne teilweise überflüssige Subcircuits,  
 $\bar{C}_v := (V(C[\text{cone}(v)]), \{(u, v) | (v, u) \in E(C[\text{cone}(v)])\})$ .

**Lemma 3.7.** *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 Kante aus  $C$ . Sei  $I$  die Menge der zu den Inputs korrespondierenden Kanten.

Es genügt zu zeigen, dass  $I$  die Kantenmenge eines gerichteten  $v$ -Schnitts in  $\bar{C}_v$  ist.

Angenommen  $I$  bildet keinen gerichteten  $v$ -Schnitt, dann existiert in  $\bar{C}_v \setminus I$  ein Weg von einem Input von  $\text{cone}(v)$  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.  $\square$

Die Menge der Matchings von  $v$  ist somit gerade die Menge aller, maximal  $\text{fanin}_{\max}$  großen  $v$ -Schnitte in  $\bar{C}_v$ , inclusive aller möglicher Inputinvertierungen und Outputinvertierungen, die als Subgraph interpretiert eine Gate der Library realisieren.

**Korollar 3.8.** *Die Anzahl der Matche eines Knotens  $v \in V(C)$  mit  $n_v := |E(C[\text{cone}(v)])|$  ist durch  $n^{\text{fanin}_{\max}} \text{fanin}_{\max}$  beschränkt*

*Beweis.* Sei  $X_v := \{E \subseteq E(C[\text{cone}(v)]) | |E| \leq \text{fanin}_{\max}\}$ . Die Menge der maximal  $\text{fanin}_{\max}$  großen  $v$ -Schnitte ist in Circuits mit beliebig vielen Highfanoutknoten durch  $|X_v|$  beschränkt. Es gilt hierbei

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

$\square$

### 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. Die Matche werden durch das iterative Hinzufügen von Gates zum aktuellen Knoten errechnet. Zu einem Knoten  $v$  gibt es maximal  $2^{|inputs(v)|} \leq 2^{fanin_{max}}$  Möglichkeiten eine Kombination von Input Gates dem Gate von  $v$  hinzuzufügen. Jeder entstehende Subcircuit (außer das Gate von  $v$  selber) besitzt, da keine Highfanoutknoten existieren, mindestens einen Inputs mehr als das Gate von  $v$ . Dieser Schritt lässt sich somit maximal  $fanin_{max}$  oft wiederholen. Daraus folgt dass in  $\mathcal{O}(2^{fanin_{max}^2})$  alle Matche für  $v$  gefunden werden können. Da  $fanin_{max}$  konstant ist, entspricht dies Linearer Laufzeit.

Bei Circuits mit beliebig vielen Highfanoutknoten, lassen sich alle Matche für einen Knoten  $v$  finden, indem man in maximal  $n_v^{fanin_{max}} fanin_{max}$  Durchläufen für alle Elemente von  $X_v$  prüft, ob diese einen  $v$ -Schnitt bilden, was in polynomieller Zeit möglich ist. Mit einem konstanten  $fanin_{max}$  ist somit das Finden von Matchen in Circuits ohne überflüssige Subcircuits und beliebig vielen Highfanoutknoten in polynomieller Laufzeit möglich. Der Einfachheit halber wird diese Laufzeit durch  $\mathcal{O}(M)$  beschrieben. **ist das mit der Konstante sinnvoll und der Fall mit überflüssigen Subcircuits ist damit leider immer noch nicht abgedeckt. gebe hier noch die Dissertation als mögliche schnellere Alternative zum Finden von Matches an muss hier noch ein kommentar hin wie es in der Praxis gelöst wird ? und soll ich diese laufzeit noch zu einer gesamtlaufzeit für alle knoten erweitern beziehungsweise die Laufzeit für die überprüfung eines Matches noch mit reinnehmen -¿ denke schon hier noch auf das finden von matchen in der heuristik eingehen !!**

## 3.6 FPTAS und Heuristik

Die Anzahl der Kandidaten ist im Allgemeinen nur exponentiell beschränkt. Der Speicheraufwand lässt sich jedoch deutlich reduzieren, wenn das Ergebnis des Algorithmus um einen Faktor  $\varepsilon > 0$  von der besten Lösung abweichen darf.

### 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  $\delta_{delay}$ , als auch in Area-Buckets der Größe  $\delta_{area}$  unterteilen. Dies wird in Abbildung 11 veranschaulicht.

Man wählt die Bucketgrößen  $\delta_{area} = \frac{\varepsilon}{2(1-\lambda)}$ ,  $\delta_{delay} = \frac{\varepsilon}{2\lambda}$  (mit Tradeoff  $\lambda$ ), und speichert erst nur den kleinsten Kandidaten in jeder Area-Bucket und

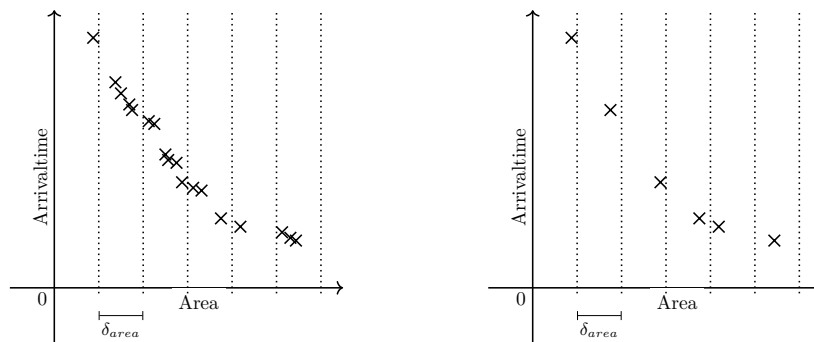


Abbildung 11: Einteilen und filtern mit Buckets

von den verbleibenden nur den schnellsten in jeder Delay-Bucket.

Dadurch existieren, bei einer maximal Zahl von Highfanoutknoten, nur beschränkt viele Kandidaten für jeden Knoten. Dies ermöglicht dem Algorithmus eine Lösung mit Kosten  $c \leq (1 + \varepsilon)OPT$ , wobei  $OPT$  die Kosten einer optimalen Lösung sind, in polynomieller Zeit zu finden. **was ist mit dem beweis ? kann ich lukas verlinken oder verlinkt der auf irgendwen ?**

### 3.6.2 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.

#### FPTAS FÜR DAS TECHNOLOGYMAPPING

**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  $c \leq (1 + \varepsilon)OPT$ .

**Laufzeit:** **hier kommt die anahme der polynomiellen laufzeit hin diese hier grob beschreiben. bei den überflüssigen Circuits kann man darauf eingehen, dass die ohne diese auch polynomiell ist man könnte dafür die überflüssigen subcircuits auch einfach vorziehen. dann wird der tiel leichter Laufzeit des FPTAS**

### 3.6.3 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 Laufzeit des FPTAS. Sie sind der Grund, warum sich der FPTAS für eine Anwendung des TechnologyMapping 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 nicht trivial, welchen man dort auswählt. Die Auswahl eines solchen Verfahrens wird 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 (und Tradeoffkurve) vorhanden ist.

---

**Algorithmus :** Heuristik für das TM mit Konvexkombination

---

**Input :** Circuit  $C$  mit finalem Output  $o$ , Library  $L$ ,  $\lambda \in [0, 1]$ ,  $\varepsilon > 0$

```

1 foreach Knoten  $v \in V(C)$  do
2   berechne Matche für  $v$ 
3   lösche alle konstante Kanten überdeckende Matche für  $v$ 
4   berechne alle nicht dominierten Kandidaten auf  $v$ 
5   if  $v$  ist Highfanoutknoten then
6      $guess \leftarrow \min_{\text{Kandidat } k \text{ auf } v} \{\lambda AT(k) + (1 - \lambda) area(k)\}$ 
7     foreach Kandidat  $k$  auf  $v$  do
8       if  $k \neq guess$  then
9         lösche  $k$ 
10  filter_kandidaten_mit_buckets( $v, \varepsilon$ )
11 Baue  $C'$ , ohne Buffer, entsprechend des besten Kandidaten auf  $o$ 
```

---

Schritt 2 berechnet evtl. nicht alle möglichen Matche. Die Routine zur Matching Suche, geht von einem Knoten aus und zieht solange Gates mit in das Match, bis  $fanin_{max}$  überschritten wird **ach scheiße mach mal eine vernünftige laufzeit für das Suchen der Matche pack das aber in Das kapitel oben drüber rein** Schritt 5-9 beschreibt ein einfaches Verfahren zur Auswahl eines Kandidaten für einen Highfanoutknoten. Angelehnt an den Kernalgorithmus wir nur der Kandidat behalten, welcher den Tradeoff minimiert. Dies garantiert keine optimale Lösung. Weitere Informationen befinden sich in Kapitel 5 und 10.

**Laufzeit:** Die Schleife aus Schritt 1 läuft durch alle Knoten. Die Laufzeit für Schritt 2 wurde bereits erläutert und Schritt 3 ist Linear in der Anzahl der gefundenen Matche implementierbar.

Schritt 4 hat einen Laufzeit von **auch hier eine Laufzeit angeben!!! kümmere dich mal um die maximale Anzahl Matches, Kandidaten und die Laufzeiten um diese zu finden und mach ein kapitel draus**

Schritt 10 ist linear in der Anzahl der Kandidaten pro Knoten und Schritt 11 linear in der Menge Kandidaten pro Knoten und  $|V(C)|$ .

Daraus folgt eine Laufzeit von **hier hin damit!**

## 4 Mehrere 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 dieser Knoten ebenfalls aus der Cone darüberliegender Knoten laufen können, werden sie ebenfalls als offene Knoten ihrer Nachfolger deklariert. Des Weiteren ist in einem Circuit mit mehreren Outputs AT alleine ein schlechtes Optimierungskriterium.

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 Geschwindigkeitsvorteile geschaffen werden können. **oder das kommt unter weitere Optimierungskriterien, dass dieses Kapitel mehrere Outputs heissen kann und die Optkriterien nicht so leer sind**

### 4.1 required Arrivaltimes

Oben 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 gewünscht 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 hier noch einmal die genaue Definition der RAT.

**Definition 4.1.** RAT:

Sei  $C$  ein Circuit und  $v \in V(C) \setminus \text{Outputs}(C)$ . Die RAT (required arrivaltime) an  $v$  ist definiert durch:

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

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

In der Praxis kommen Signal 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 Interessanter einen gegebenen Circuit hinsichtlich des negativen Slacks zu verbessern. Hieraus ergeben 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.

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

*hier passt sehr gut rein Lukas FPTAS zu erwähnen da er auf den schlechtesten Pfad angewendet wird, war die anzahl der Highfanout gates vorhersehbar? -> in den nächsten absatz mit einbauen*

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 hintereingeschalteten 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(*schon definiert(wenn nein nötig?)*), umformen. Dieser lässt sich dann mit Delay optimierenden TechnologyMapping (in polynomieller Laufzeit*zeigen ?*) umbauen und wieder in den Chip einbauen. Der Große Vorteil von diesem Vorgehen sind überschaubar große Instanzen und eine Beschleunigung des gesamten Chips in sehr schneller Zeit. 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 TechnologyMapping 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 Mehrere Outputs

Wie in der Einleitung beschrieben, ist es das Ziel dieser Arbeit eine Heuristik für das TechnologyMapping zu entwickeln, welche auf großen Teilen eines Chips lauffähig (bezüglich Laufzeit) ist. Da ein solcher Chip mehr als nur einen Output-Pin besitzt, lässt er sich in zusammenhängende Circuits unterteilen, welche mehr als einen Output-Knoten besitzen. Folgende Umbauten sind notwendig um mit den 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, sodass diese sich an den sich überschneidenden Knoten gleichen. Somit lässt sich ein Circuit mit den bekannten Mitteln ein Circuit konstruieren, welcher eine Kostenfunktion hinsichtlich Größe und WS optimiert.

Die zweite Änderung hat sich dadurch bereits angekündigt. 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, jedoch ein anderer Output könnte durch diesen Umbau schlechter werden.

Um dieses Problem zu umgehen, verändert man  $C$  vor dem TechnologyMapping 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 beschriebene Problem gelöst.

Wie bereits in Kapitel 4.1 erwähnt 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 TechnologyMapping noch alle Outputs, mit der bis zu ihnen realisierten Logischen Funktion, vorhanden sein. Daraus folgt, dass über einen Output-Knoten, welcher in dem Circuit noch mindestens einen Nachfolger hat, nicht gematcht werden darf, denn sonst würde



ein nicht erlaubter Seitenoutput entstehen.

Die lässt dadurch bewerkstelligen, dass man eine seiner ausgehenden Kanten als konstant deklariert, wie das bereits bei den zu langen Kanten geschehen ist.

indem unterkapitel noch mehr bilder bzw schönerer aufbau ist nämlich aktuell viel text!!

## 5 Premapping von Highfanoutknoten

Der exponentielle Anstieg der Kandidatenmenge wird, wie oben gezeigt, durch die Highfanout-Knoten verursacht. Daraus folgt, dass ein sehr großes Laufzeit Potenzial in der Reduzierung der Kandidaten für diese Knoten liegt. Die Kandidatenmenge eines jeden Highfanoutknotens wird, wie bereits in Kapitel 3.6.3 geschehen, auf eins reduziert. Diese Routine wird auch das Premapping der Highfanoutknoten genannt. Dadurch 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 die einzige verbleibende Tradeoffkurve nach dem Kandidaten mit den geringsten Kosten gesucht wird. Dadurch ist das Finden des noch bestmöglichen Circuit Kandidaten 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 der Heuristik.

Im folgenden werden drei verschiedene Routinen vorgestellt und auf ihre Eigenschaften eingegangen. Genauere Informationen über die Unterschiede der Resultate dieser Routinen finden sich in Kapitel 10.

### 5.1 triviales Premapping

Diese Methode des Premappings wurde bereits in Kapitel 3.6.3 benutzt. 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 liegt nahe, lässt sich jedoch noch weiter verbessern.

Die garantiert optimale Lösung findet das triviale Premapping jedoch nur mit Tradeoff  $\lambda \in \{0, 1\}$

## 5.2 erweitertes Premapping

gilt es noch zu entwickeln

## 5.3 Premapping durch Schätzen

Beim Premapping durch schätzen wird versucht eine Vermutung für die Kosten eines guten Kandidaten aufzustellen. Ausgewählt wird dann der Kandidat, welcher die geringste Differenz zu den vermuteten Kosten besitzt.

Die Schätzung erfolgt durch zwei TechnologyMapping Läufe. Das TechnologyMapping wird mit demselben Circuit und den Tradeoff Parametern 0, 1 gestartet. das ist eine gute abschätzung weil man durch die langsame implementierung des kleinen circuits kosten in der Rat bekommt plus die Methode aus Lukas heuristik entnehmen und dazupacken

## 6 Präprozessing

Die Möglichkeiten des Matchings 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 möglich vor dem TechnologyMapping Algorithmus jedes Gate mit mehr als zwei eingehenden Inputs 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 (beweis verlinken ? und auf jedem realen Chip standartmäßig 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 TechnologyMapping deutlich erweitert werden. Jedoch werden für ein AND4 Gate Beispielsweise 3 AND2 Gates eingesetzt, was dazu führt, dass sich im Allgemeinen die Kosten des Ausgangscircuits verschlechtern.

Eine ausführliche Analyse der Vor- und Nachteile des decomposen finden sich in Kappitel 10.

## 7 Weitere Optimierungskriterien

Das TechnologyMapping arbeitet im Chip-Design auf realen Instanzen. Dadurch kommen, zu den bereits vorgestellten, weitere Optimierungskriterien hinzu. Es handelt sich hierbei einmal um Ressourcen, welche in die Kostenfunktion mit eingebracht werden können und somit während des TechnologyMapping optimiert werden. Des Weiteren verursacht der neu implementierte Circuit Kosten welche im TechnologyMapping 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.

## 7.1 pinabhängiges Delay

Bis zu diesem Punkt 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. Physikalisch werden die Signale der Inputs zwar alle miteinander verrechnet, jedoch geschieht dies nicht gleichzeitig und somit müssen nicht alle Signale zur selben Zeit an den Inputs anliegen.

Die spätere Ankunftszeit lässt sich durch einen kleineren Delaywert, spezifisch für diesen Input, realisieren. Denn wenn das Signal schneller durch das Gate gelangen kann, so brauchst 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 TechnologyMapping 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 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 Optimalität des, noch vorzustellenden, allgemeinen Algorithmus zu wahren, 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.

## 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 Benutzung abhängt. Dieser variable Energieverbrauch ist durch die dynamic Power beschrieben.

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 Circuits 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 lässt 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 der drei Kriterien zusammen verzichtet.

In Kapitel 10 finden sich weitere Informationen über die Auswirkungen von diesem Austausch in der Kostenfunktion.

### 7.3 Layer Assignment

Die Knoten eines Circuits sind durch Kanten miteinander verbunden. Diese müssen auf dem Chip in einer Menge von Schichten(Layer) liegen. Hierbei besitzt jede Schicht spezifische Eigenschaften an Geschwindigkeit und anderweitigen Kosten.

Während jedes TechnologyMapping Algorithmus kommen neue Kanten hinzu oder verschwinden in dem äquivalenten Circuit. Das zu lösende Optimierungsproblem ist es nun den neu hinzugekommenen Kanten Layer zuzuordnen. In den Implementierungen der Algorithmen dieser Arbeit, werden die Kanten auf die Layer gelegt, auf welchen Sie vorher lagen.

Es steht noch aus zu prüfen ob und wie dieses Verfahren verbessert werden kann. Da dies bisher die beste Methode diese Problem 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 eine weitere Erwähnung in dieser Arbeit finden.

## 8 Version der Heuristik, welche obige Kriterien beherzigt

## 9 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 10 nicht darauf eingegangen.

Es folgt die allgemeine Definierung 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 seine 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  $\forall c \in \mathcal{C} : g_c : \mathcal{B}_c \rightarrow \mathbb{R}^{\mathcal{R}_+}$ .

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  $g_c : \mathcal{B}_c \rightarrow \mathbb{R}_+^{\mathcal{R}} \forall c \in \mathcal{C}$

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

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

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 TechnologyMapping ansatz! theoretische Beweise?

## 10 Laufzeitanalyse

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

### 10.1 Struktur realer Instanzen

Die Instanzen eines Chips sind im Folgenden alle maximal zusammenhängenden, mit TechnologyMapping 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 TechnologyMapping 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 wurde auf dem Stand desgleichen Snapshots bearbeitet. Dadurch lassen sich die Verbesserungen von Instanzen durch die Algorithmen miteinander vergleichen, auch wenn Sie von verschiedenen Chips stammen.

Die Folgenden Angaben betreffen den Mittelwert aller getesteter Instanzen.  
**anzahl getesteter instanzen**

- Anteil Highfanouknoten
- Anteil Outputs / Anteil outputs mit Nachfolger
- Anteil an Inputs

Abbildung ?? 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

### 10.2 Analyse der Ergebnisse

Funktion/Abbildung mit Tradeoff in bezug zur Kostenersparnis  
irgendwo muss noch erwähnt werden, dass die größe/vt und at werte am anfang gleich gesetzt werden

### 10.3 Laufzeitanalyse

Aus der theoretischen Laufzeitschranke aus Kapitel 3.6.3 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, abzüglich 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 de komplexen Gates grundsätzlich erlaubt, einer der zwei Gruppen zuordnen. Diese Unterscheidung findet sich ebenfalls in Abbildung ??.

## 11 Fazit und Ausblick