Heuristiken für das TechnologyMapping

Alexander Zorn

Geboren am 26. Mai 1996 in Bonn

26. Juni 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

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 gegeben 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.

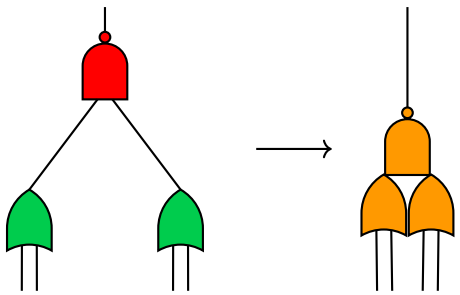

Abbildung 1: Zwei Realisierungen der Logischen Funktion $\neg((w \vee x) \wedge (y \vee z))$

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

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

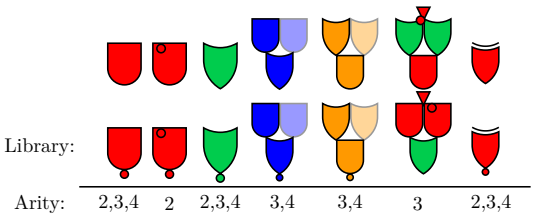

Abbildung 2: Beispiel einer Library

$area_g$ 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 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 kreisfeier 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 $f : X \to inputs(p)$
- Ein Funktion $inv : inputs(p) \to \{not\_inv, inv\}$

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.

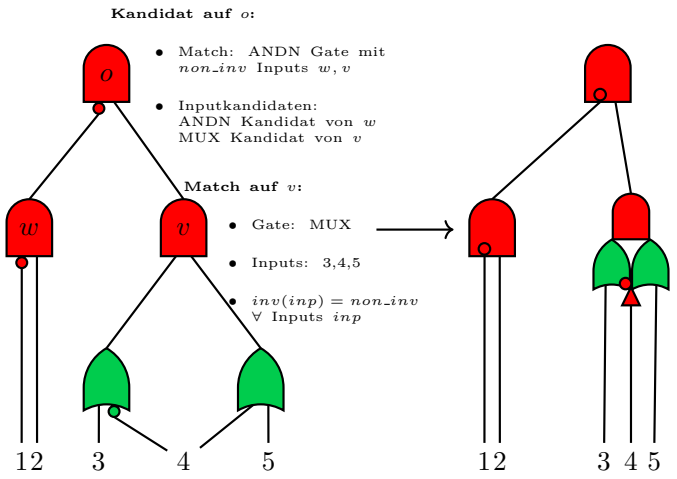

Abbildung 3: Beispiel einer Library

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:

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

Wobei $can(K)$ die Menge der Kandidaten von K 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 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]
```

Definition 2.7. optimaler TechnologyMapping Algorithmus:
Ein Algorithmus für das TechnologyMapping auf einem Circuit C heißt optimal, wenn er den (bzgl. der Kostenfunktion) besten äquvalenten 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 erlaubte Operationen das logische äquvalente matchen von Subcircuits mit Elementen der Library.

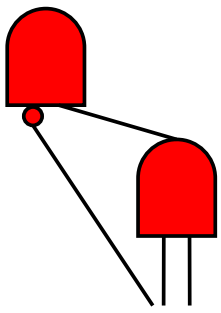

Abbildung 4: Ein Circuit desen boolesche Funktion $f = 0$ ist

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

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 Kandidate, 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 2.9. *Der Algorithmus für das (einfache) TechnologyMapping 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. 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 $\mathcal{O}(|V(C)|^3|L|^2)$. □

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 äquvalent 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.

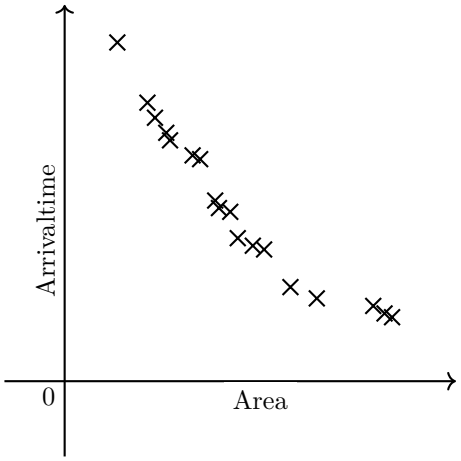

Abbildung 5: Ordnen der Kandidaten in einer Tradeoffkurve

Die beiden vorgestellten Probleme sind NP-vollständig. Daraus folgt, dass sich ab diesem Punkt wahrscheinlich kein polynomieller optimaler Algorithmus für das Technology-Mapping 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 Paper einsetzen wieder. Abbildung 6 verbildlicht 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?

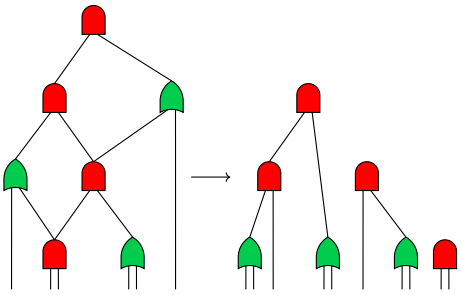

Abbildung 6: Unterteilen eines Circuit in Highfanoutfreie Subcircuits

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

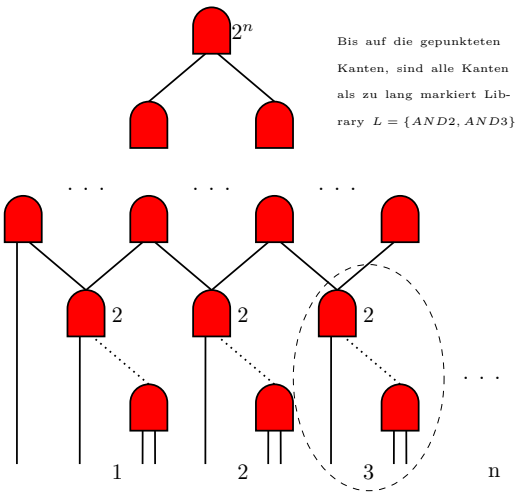

Abbildung 7: Exponentiell viele Kandidaten bereits bei sehr eingeschränkter Library

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:

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

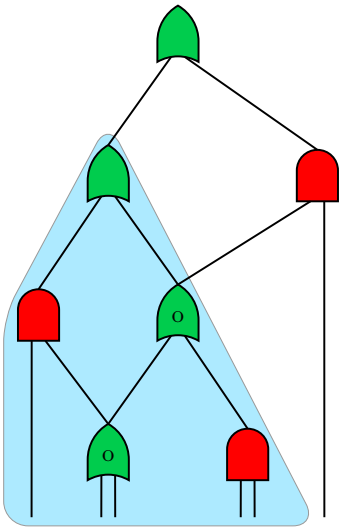

Abbildung 8: Visualisierung der Definitionen 3.2 bis 3.4

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.

3.3 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

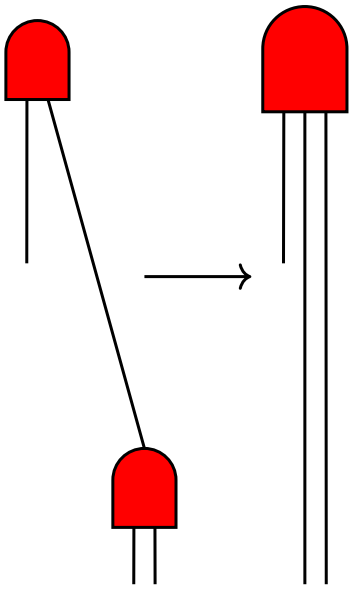

Abbildung 9: Visualisierung eines Matches über eine lange Kante

Klasse von Kanten eingeführt über welche man nicht matchen darf. Diese Kanten bezeichet 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.

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 $f : \{0,1\}^n \to \{0,1\}^m$. 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 bedeuted, 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

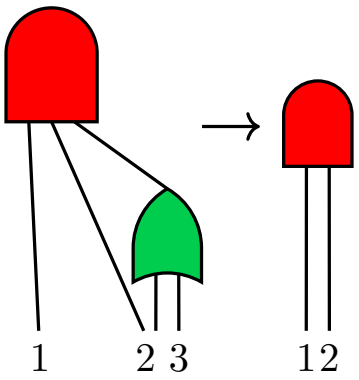

Abbildung 10: Nur Input 1 und 2 sind die relevanten Inputs

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.was ist mit dem Laufzeit statistiken?
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

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

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 Kantemenge K aus C. Es ist möglich, dass $|K| > 1$, denn ein Match kann mehrere ausgehende Kanten eines Knoten zu einer Kante zusammenfasst. Dies veranschaulicht Abbildung ?? für ein Match eines MUX Gates. Sei I die Menge der zu den Inputs korrespondierenden Kantenmengen.
Es genügt zu zeigen, dass alle Kanten I' in I gerade die Kantenmenge eines gerichteten v-Schnitts in $\bar{C}_v$ ist.
Angenommen I' bildet keinen gerichteten v-Schnitt, dann existiert in $\bar{C}_v \backslash I$ ein Weg von einen 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. Die wird ebenfalls durch Abbildung ?? veranschaulicht. Ohne Highfanoutknoten ist diese Zuweisung jedoch, abgesehen von den möglichen Invertierungen der Inputs und des Outputs, eindeutig, wenn es keine zwei Gate 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 $\bar{C}_v$, inclusive aller möglicher Inputinvertierungen und Outputinvertierungen, die als Subgraph interpretiert ein Gate der Library realisieren.

Korollar 3.8. *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 noch zwei Verschiedene mögliche Output-Invertierungen und maximal $2^{fanin_{max}}$ viele Möglichen 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 3.9. *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 3.8 gibt eine obere Schranke für die Anzahl aller Matche inklusive der möglichen Invertierungen an. Sei $u \in x \in X_v$ (Definition siehe oben) zusätzlich 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 3.8 folgt die Aussage. □

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

Korollar 3.10. *Sei C ein Circuit ohne Highfanoutknoten. Die Anzahl der Matche eines Knoten ist durch $2^{2fanin_{max}+1} fanin_{max}$ beschränkt.*

Beweis. Ausgehen von einem Knoten $v \in V(C)$ mit maximal $fanin_{max}$ Inputs, gibt es $2^{fanin_{max}}$ Möglichkeiten die an den Input liegenden Gates mit in das Match einzuschließen. Abgesehen von einer, 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 jedes 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 3.11. 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 3.10 folgt, dass potenziellen 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 $\mathcal{O}(|V(C)|^2|L|)$ und somit polynomiell, möglich ist.

Bei Circuits mit beliebig vielen Highfanoutknoten aber beschränktem $fanout_{max}$, lassen sich durch Korollar 3.9 und gleichem Vorgehen wie oben 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

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 3.12. *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 Knoten 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 11 veranschaulicht.
Man wählt die Bucketgrößen $\delta_{area} = \frac{\varepsilon}{2(1-\lambda)}, \delta_{delay} = \frac{\varepsilon}{2\lambda}$ (mit Tradeoff λ),

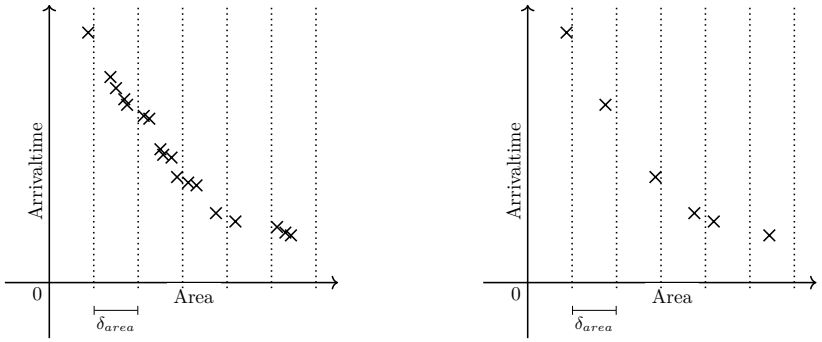

Abbildung 11: Einteilen und filtern mit Buckets

und speichert erst nur den kleinsten Kandidaten in jeder Area-Bucket und von den verbleibenden nur den schnellsten in jeder Delay-Bucket.
Dadurch ist die maximale Anzahl an Kandidaten in einer Tradeoffkurve polynomiell beschränkt.

Definition 3.13. 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.

M genauer einschränen und die Schranken irgendwo mit hineinbringen bzw beweisen, dass M pol. beschränkt ist. was ist mit dem beweis ? kann ich lukas verlinken oder verlinkt der auf irgendwen ?

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 die 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.

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

algorithmus und laufzeit einfügen

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

3.8 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 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 Kandidate 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, ε)
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 Kandidate 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 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, wer-

den sie ebenfalls als offene Knoten ihrer Nachfolger deklariert. Des Weiteren ist in einem Ciruit 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 Geschwindigleitsvorteile 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 nocheinmal 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 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 vorhersebar? -¿ 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 hintereinangeschalteten Kette von Knoten besteht. Dieser lässt sich nun mit geeigneten Algorithmen füge hier mal ein Beispiel oder einen Verweis an zu einem äquvalenten Circuit C', mit geringerer Tiefe(schon definiert(wenn nein nötig?), umformen. Dieser lässt sich dann mit Delay optimierenden TechnologyMapping (in polynomieller Laufzeitzeigen ?) 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 **??** 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 Kandidate 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 **??** 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 Augangscircuits verschlechtern.
Eine ausfürhliche 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 eine 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 Ankunftsmöglichkeit 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 Dealy des Outputs verbessern. Aus diesem Grund ändert sich die AT eines Knotens wiefolgt:

Definition 7.1. AT mit pinabhängigen Delay:
Sei C ein Circuit und $v \in V(C)$. Die AT von v mit pinabhängigen Delay ist wiefolgt 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 Benutzung abhängt. Dieser variable Energieverbrauch ist durch die dxnamic 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. 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 TechnologyMapping 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, sodass die Terminalmengen untereinander auf dem Chip verbunden werden können und die zusätzlichen Kosten nicht beliebig 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 steht noch aus zu überprüfen, 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 das Problem an dieser Stelle zu lösen ist 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 keine 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 $\forall c \in \mathcal{C} : g_c : \mathcal{B}_c \to \mathbb{R}^{\mathcal{R}+}$.
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 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 gestesteten Chips wurde auf dem Stand desgleichen Snapshots bearbeitet. Dadurch lassen sich die Verbesserungen von Instanzen durch die Algorithmen miteinenander 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

irgendwo muss noch erwähnt werden, dass die größe/vt und at werte am anfang gleich gesetzt werden

10.2.1 Tradeoffparameter

10.2.2 Premapping

10.2.3 Präprozessing

inclusive der graphen größe vergleiche

10.2.4 Power und Area Vergleich

10.2.5 Bucket filetering ε im vergleich

10.2.6 Gütevergleich kleiner öptimal "gel instanzen

10.2.7 Verhalten weiterer Kosten

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

10.2.8 Zusammenfassung?

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

10.3 Laufzeitanalyse

Aus der theoretischen Laufzeitschranke aus Kapitel **??** 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 ??.

10.3.1 globale Laufzeitanalyse

welche größenordnungen sind überhaupt lösbar

10.3.2 locale 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 ?

10.3.3 Bucket filetering ε im vergleich

10.3.4 Laufzeitverlgeich kleiner öptimal "gel instanzen

10.4 Güte laufzeit vergleich

10.4.1 kleine Instanzen

10.4.2 allgemein

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

10.4.3 Güte Bucket filetering ε

11 Fazit und Ausblick