# Heuristiken für das TechnologyMapping

Alexander Zorn

Geboren am 26. Mai 1996 in Bonn

31. März 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

# 1 Einleitung

# 2 Terminologie & grundlegender Algorithmus

Das Technology Mapping ist ein Problem, einen gegebenen Circuit (Subjectgraph) unter gegebener Library (Patterngraphs) logisch äquivalent umzubauen, sodass er eine gewisse Zielfunktion (z.B. Arrivaltime oder Area) optimiert. Um dies zu präzisieren und eine grundlegende Terminologie für die späteren Algorithmen zu haben, folgen einige Definitionen.

**Definition 2.1.** Library: Eine Library ist eine Menge $L$ von Gates (boolsche Funktionen) mit zwei Abbildungen $d, \text{area} : L \to \mathcal{R}_{\geq 0}$, die jedem Gate sowohl eine Verzögerung $d_l$, als auch eine Fläche $\text{area}_l$ zuordnen.

**Definition 2.2.** Circuit: Ein Circuit $C$ auf der Library $L$ ist ein zusammenhängender gerichteter azyklischer Graph (DAG), bei dem jeder Knoten einer dieser 3 Arten entspricht:

- einem Inputknoten ohne eingehende Kanten
- einem Gate aus L mit $\geq 1$ eingehenden und $\geq 1$ ausgehenden Kanten
- einem Outputknoten ohne ausgehende Kanten

Jeder Gateknoten kann an jeder seiner eingehenden Kanten einen Inverter vorschalten. Für einen Knoten $v$ sei $\text{fanin}(v)$ die Zahl seiner eingehenden, $\text{fanout}(v)$ die Zahl seiner ausgehenden Kanten. Knoten mit $\text{fanout}(v) > 1$ heißen Highfanoutknoten. Wir betrachten vorerst nur Circuits mit exakt einem Outputknoten.

**Definition 2.3.** cone: Für einen Knoten g aus einem DAG S bezeichne

$$cone\,(g) := S\left[V \cup \{g\}\right], V = \{v \in V(S) : \exists v\text{-}g\text{-Weg in } S\}$$

Sowie für eine Knotenmenge G sei $cone(G) := S\left[\cup_{g \in G} V(cone(g))\right]$

**Algorithmus :** TechnologyMapping auf einer Arboreszens

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

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