HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES

Fraunhofer IIS

DIPLOMARBEIT

Frau
Franziska Heinicke

Untersuchung von heuristischen Optimierungsverfahren zur hierarchischen Platzierung von Standardzellen in 3D-Aufbauten

2010

Fakultät: **Mathematik/Naturwissenschaften/Informatik**

Diplomarbeit

Untersuchung von heuristischen Optimierungsverfahren zur hierarchischen Platzierung von Standardzellen in 3D-Aufbauten

Autor:
Franziska Heinicke

Studiengang:
Angewandte Mathematik

Seminargruppe:
MA06w1

Erstprüfer:
Prof. Dr. rer. nat. Peter Tittmann

Zweitprüfer:
Dipl. Inf. Andy Heinig

Mittweida, Dezember 2010

Bibliographische Angaben

Heinicke, Franziska:
Untersuchung von heuristischen Optimierungsverfahren zur hierarchischen Platzierung von Standardzellen in 3D-Aufbauten – Dezember 2010, 111 Seiten, 71 Abbildungen, 10 Tabellen.
Mittweida, Hochschule Mittweida (FH), University of Applied Sciences, Fakultät Mathematik/Naturwissenschaften/Informatik, Diplomarbeit, 2010

Referat:

Das Ziel der Diplomarbeit ist es, eine hierarchische Methodik für die Erstellung eines 3D-Aufbaus eines Chips zu entwickeln. Dabei sollen insbesondere die heuristischen Optimierungsverfahren Simulated Annealing, Threshold Accepting und Sintflutalgorithmus bezüglich ihrer Eignung miteinander verglichen werden. Die hierarchische Herangehensweise wird durch eine Partitionierung des Schaltkreises mit dem Verfahren hMETIS in größere, zu platzierende Superzellen realisiert. Die Platzierung erfolgt mit der Datenstruktur Sequenz-Paar angewendet auf mehreren Ebenen und den bereits genannten heuristischen Optimierungsverfahren.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich während meines Studiums und beim Schreiben der Diplomarbeit unterstützt haben.

Die Bearbeitung meiner Diplomarbeit erfolgte am Fraunhofer Institut für Integrierte Schaltung. Dabei stand mir der Betreuer der Diplomarbeit Herr Heinig immer tatkräftig zur Seite.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Integrierte Schaltung

Wir leben in einer Zeit, in der fast alles voll automatisierbar ist. Wir können die Kaffeemaschine so programmieren, dass sie morgens halb sieben zwei Tassen Kaffee kocht, können per Knopfdruck die Temperatur im Auto regulieren und mit dem Handy den Tag organisieren.
All das ist möglich, weil wir heute die Fähigkeit haben, höchst komplexe Schaltungen auf einer sehr kleinen Fläche in sogenannten Chips unterzubringen. Doch was ist überhaupt ein Chip?
Ein Chip ist ein Träger, auf dem eine elektrische Schaltung aufgebracht wurde. Diese Schaltung wird dann Integrierte Schaltung oder abgekürzt IC (Integrated Circuit) genannt [Alb07]. Die Beschreibung der Realisierung der Schaltung auf dem Träger – also der Art, Größe und Platzierung der elektrischen Bauteile und der Leitungen – wird als Layout bezeichnet [Kra94].
Ein Chip besteht grundsätzlich aus drei Schichten: dem Träger, der Schicht der Zellen und der Verdrahtungsschicht. Das Trägermaterial wird im Allgemeinen als Substrat bezeichnet und besteht zumeist aus Silizium. Die Transistoren werden in das Substrat integriert, darüber werden mehrere Metallschichten für die Verdrahtung aufgetragen [Her02]. Es werden mehrere Transistoren, die eine logische Einheit bilden, zu Zellen zusammengefasst. Elektrische Verbindungen zwischen Leitungen benachbarter Metallschichten werden Vias (Vertical Interconnect Access) oder auch Durchkontaktierung genannt.
In Abbildung 1.1 wird der Chip-Aufbau durch eine schematische Darstellung des Querschnitts verdeutlicht.
Es ist üblich, die Leitungen einer Metallschicht bevorzugt parallel in einer Richtung anzuordnen – abwechselnd eine Metallschicht mit horizontal und eine mit vertikal

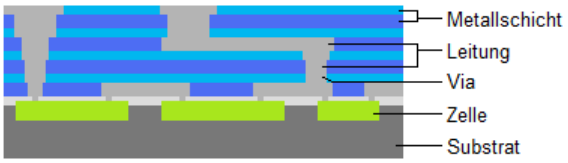

Abb. 1.1: Schematische Darstellung des Querschnitts eines Chips

positionierten Leitungen – damit möglichst viele Leitungen platziert werden können. Schräge Leitungen werden zumeist nicht in Betracht gezogen, da sich dadurch die Komplexität des Verdrahtungsproblems erhöht.
Die Anordnung der Zellen auf der Chipfläche ist gewissen Regeln unterworfen [Sch10], so sollten die Zellen, die eine elektrische Verbindung nach außen haben (welche als IOs bezeichnet werden) am Rand des Chips liegen. Die Chipfläche sollte im Allgemeinen rechteckig sein und es dürfen sich keine Zellen überlappen. Wie in Abbildung 1.2 zu sehen, wird die Chipfläche in zwei Bereiche unterteilt: den IO-Bereich und die Core Area. Der IO-Bereich ist ein Ring um die Core Area, auf dem die IOs liegen, damit sie von außen ankontaktiert werden können. In der Core Area befinden sich die logischen Zellen und die Leitungen.

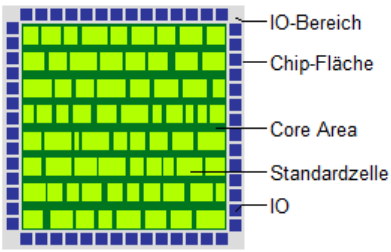

Abb. 1.2: Flächeneinteilung eines Chips

Heutzutage werden die IOs häufig auch innerhalb der Chipfläche untergebracht, was vor allem bei Schaltungen mit vielen IOs notwendig ist. Darauf soll hier aber nicht näher eingegangen werden, da die Platzierung der IOs nicht Teil der Aufgabenstellung ist.
Es gibt verschiedene Typen von Zellen, die in einem Chip Verwendung finden können. Zu den beiden wichtigsten gehören die Makro- und die Standardzellen. Makrozellen sind sehr groß und haben meist eine komplexe Aufgabe, Standardzellen hingegen führen einfache logische Funktionen aus [Kra94]. In dieser Arbeit sollen nur Schaltungen betrachtet werden, deren Zellen ausschließlich Standardzellen sind. Diese haben den Vorteil, dass sie alle dieselbe Höhe haben und somit in Reihen nebeneinander

platziert werden können. Damit vereinfacht sich das Platzierungsproblem erheblich. Ein wichtiger Punkt in der Layout-Entwicklung ist die Einhaltung der Zeitbedingung [Unb09]. Jedes Signal muss innerhalb einer vorgegebenen Zeit den Chip passiert haben. Diese Zeitspanne wird der Takt des Chips genannt, in dem Berechnungen ausgeführt werden können. Da der Chip später einmal in ein komplexes System, wie zum Beispiel einen Computer, integriert werden soll, ist der Takt von außen vorgegeben und muss eingehalten werden, damit der Chip kompatibel ist. Verschiedene Signale nehmen verschiedene Wege innerhalb des Chips und haben abhängig davon unterschiedliche Zellen zu passieren. Jede Zelle benötigt Zeit zur Verarbeitung des Signals, aber auch die Leitungen dazwischen verzögern das Signal. Der Zeitverzug, der durch die Leitungen verursacht wird, hängt in etwa quadratisch von deren Länge ab [HJR09].
Lange Leitungen verzögern das Signal aber nicht nur, sie schwächen es auch durch den elektrischen Widerstand und besonders stark durch kapazitive Effekte ab. Deswegen kann es nötig sein, eine andere, stärkere und damit größere Zelle einzusetzen oder das Signal mit einer zusätzlichen Zelle zu verstärken.
Aufgabe der Layout-Entwicklung ist also nicht nur die Platzierung der Zellen und Leitungen, sondern auch das Anpassen des elektrischen Schaltkreises an die Gegebenheiten, die aus der Platzierung resultieren. Dabei bleibt die Funktion der Schaltung erhalten, es werden nur Veränderungen vorgenommen, die das Signal an kritischen Stellen verstärken und damit die Übertragung sichern [Sch10].

1.2 3D-Integration

Seit Jahren findet eine stetige Steigerung in der Chip-Technologie statt, es werden immer komplexere Schaltungen auf vergleichsweise kleinen Chips integriert. Ein Chip hatte in den frühen Siebzigern keine zehntausend Transistoren – heute besitzen sie mehrere Milliarden. Wäre die Größe der Chips äquivalent gestiegen, gäbe es keine Handys und Laptops, da sie viel zu groß und schwer wären. Da aber der Herstellungsprozess immer präziser wurde und die einzelnen Bauteile damit immer kleiner gefertigt werden konnten, werden heute schon höchst komplexe Systeme auf einer Fläche von wenigen Quadratmillimetern untergebracht.
Die technischen Fähigkeiten in der Fertigung allein reichen aber nicht, um so große

Schaltungen in einen Chip zu integrieren. Es müssen Verfahren zur automatisierten Erstellung des Layouts entwickelt werden, welche die technologischen Besonderheiten berücksichtigen und mit möglichst wenig Aufwand eine gute Anordnung der Zellen und Leitungen berechnen. Für zweidimensionale Chips, also ICs, die nur auf einer Ebene aufgebracht sind, gibt es kommerzielle Tools für diese Berechnungen.
Heute sind die Schaltungen aber teilweise so groß, dass ein zweidimensionales Layout nicht mehr möglich ist. Eine Ursache dafür ist die zunehmende Länge der Leitungen mit den daraus resultierenden Problemen, wie dem Abfall der Signalleistung und der Zeitverzögerung, welche bereits in Abschnitt 1.1 näher erläutert wurden.

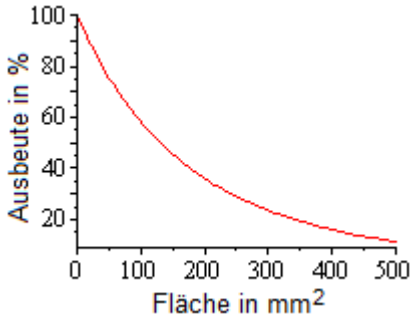

Abb. 1.3: Ausbeute an funktionsfähigen Chips in Abhängigkeit von der Fläche

Mit zunehmender Fläche steigt außerdem der Anteil an defekten Chips stark an. In Abbildung 1.3 ist der Prozentsatz an fehlerfrei produzierten Chips in Abhängigkeit von der Chipfläche zu sehen. Dieser wurde mit dem Ausbeute-Modell aus Kapitel 5.2 berechnet.
Ein Ausweg wird in der Aufteilung des ICs auf mehrere Ebenen gesehen, weil dabei kleinere Flächen entstehen und somit der Anteil an fehlerhaften Chips sinkt. Außerdem liegen die Zellen bei einer dreidimensionalen Anordnung dichter beieinander, wodurch die Verbindungslängen verkürzt werden [KAL09].

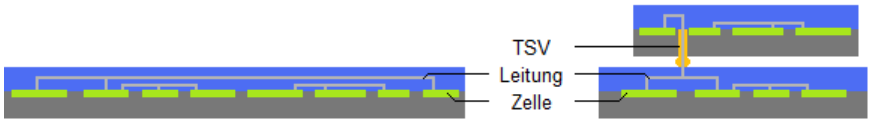

Abb. 1.4: Kürzere Leitungen im 3D-Chip

In Abbildung 1.4 ist eine Leitung dargestellt, die durch den 3D-Aufbau des Chips verkürzt wird.
Um einen 3D-Chip herzustellen, werden im Prinzip mehrere Chips gefertigt, die alle

einen Teil des elektrischen Schaltkreises enthalten und anschließend übereinander gestapelt werden. Dabei sind alle IOs auf einer Ebene zu platzieren. Die elektrischen Verbindungen zwischen Zellen auf verschiedenen Ebenen des Chips werden mit TSVs[1] (Throught Silicon Vias) ermöglicht. In Abbildung 1.5 ist ein solcher Aufbau dargestellt.

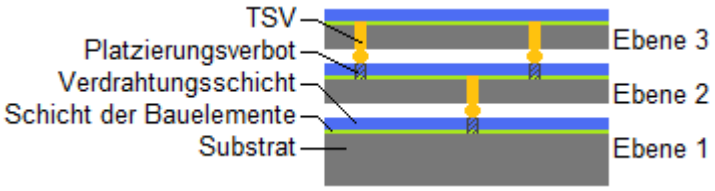

Abb. 1.5: Schematische Darstellung des 3D-Aufbaus eines Chips

Heutzutage werden die TSVs in verschiedenen Größen gefertigt, in dieser Arbeit wurde eine Seitenlänge von jeweils 3µm angenommen. Damit haben die TSVs ungefähr die doppelte Fläche der Standardzellen in einer üblichen 65nm-Technologie. Um ein Brechen des Trägers zu verhindern, dürfen die TSVs nicht zu dicht nebeneinander platziert werden. Deswegen muss zwischen diesen ein hinreichend großer Abstand eingehalten werden[2]. Obwohl dazwischen Zellen und Leitungen integriert werden können, nehmen die TSVs viel Platz in Anspruch. Es ist also wichtig, so wenig TSVs wie möglich einzusetzen und diese den Vorgaben entsprechend zu platzieren.

1.3 Aufgabenstellung

Für die Entwicklung von 2D-Chips – also Chips, deren Zellen auf nur einer Ebene aufgebracht sind – gibt es einige Platzierungs- und Verdrahtungstools (bezeichnet als P&R-Tool für engl. Place & Route) für die Berechnung des Layouts. Solche Tools sind für 3D-Aufbauten noch zu entwickeln.
Ziel dieser Diplomarbeit ist die Entwicklung einer Methode, die einen elektrischen Schaltkreis auf mehrere Ebenen kostenoptimal aufteilt und die TSVs einfügt, um im Anschluss das Layout der Ebenen einzeln mit einem herkömmlichen P&R-Tool zu

[1] Um eine elektrische Verbindung zwischen zwei Ebenen herzustellen, werden Kanäle in den Träger geätzt und mit einem leitfähigen Material gefüllt. Diese Kanäle werden als TSVs bezeichnet. Die TSVs werden durch eine Lötkugel mit der darunter liegenden Ebene verbunden. Unter der Lötkugel dürfen keine Zellen und Leitungen platziert werden - dort ist eine Freifläche zu reservieren.
[2] Wie groß dieser Abstand sein muss, ist noch nicht hinreichend erforscht. In dieser Arbeit wurde ein Mindestabstand von 3µm angenommen.

erstellen. Diese Methode soll hierarchisch vorgehen, also den elektrischen Schaltkreis vorher durch die Fusion von mehreren Zellen zu Superzellen verkleinern.

Abbildung 1.6 zeigt den grundsätzlichen Ablauf der in dieser Arbeit vorgestellten Methode. Die einzelnen Punkte des Programms werden im Folgenden kurz erläutert.

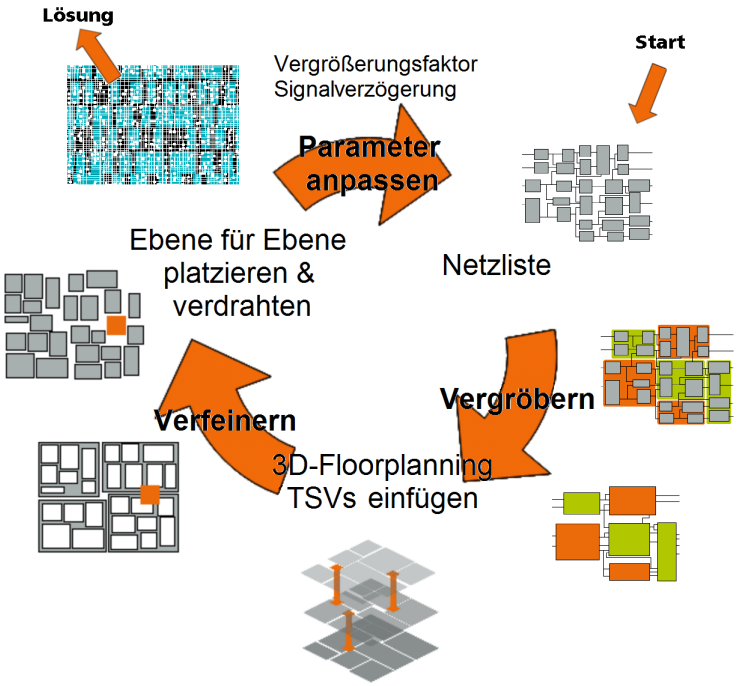

Abb. 1.6: Ablauf der Entwicklung eines 3D-Layouts

Netzliste und Parameter

Das Programm erhält als Eingabe die Netzliste des Schaltkreises. Darin sind alle wichtigen Informationen enthalten: Es wird für jede Zelle angegeben, welche Leitungen anzuschließen sind, wie groß die Zelle ist und in welcher Zeit ein Signal die Zelle passiert. Des Weiteren ist der Takt vorgegeben.

In dem Programm gibt es verschiedene Parameter, die abhängig von der Schaltung und den Ergebnissen aus vorherigen Berechnungen angepasst werden sollen. Zu diesen Parametern gehört unter anderem ein Vergrößerungsfaktor für die Größe der beim Vergröbern entstandenen Superzellen und eine Abschätzung für die Länge der lokalen Leitungen[3], sowie die durchschnittliche Verzögerung des Signals in den Leitungen pro Längeneinheit. Diese Parameter werden im ersten Durchlauf auf einen Standardwert

[3] Lokale Leitungen sind sehr kurze Leitungen. Alle Leitungen innerhalb einer Superzelle werden im Folgenden als lokale Leitungen bezeichnet.

gesetzt und vor jedem weiteren Durchlauf abhängig von den Ergebnissen angepasst. Wurde also zum Beispiel die Zeitbedingung in der Abschätzung des Programms eingehalten, aber mit dem P&R-Tool keine Platzierung für die einzelnen Ebenen gefunden, welche die Zeitbedingung einhält, so sollte die Signalverzögerung der Leitung pro Längeneinheit und die Abschätzung für die Länge der lokalen Leitungen erhöht werden.

Vergröbern

Da die Netzliste mehrere hunderttausend Zellen enthält, ist es sinnvoll, vorher mehrere Zellen zusammenzufassen und als Einheit zu betrachten. Der Vorteil liegt vor allem in der Zeitersparnis. Deswegen werden zunächst mehrere Zellen zu hundert bis fünfhundert Superzellen fusioniert. Wie dies genau geschieht, wird in Kapitel 3 erläutert. Die Größe einer Superzelle hängt dabei nicht nur von den Zellen ab, die in ihr liegen, sondern auch von den bereits erwähnten Parametern und der Anzahl an lokalen Leitungen in der Superzelle. Um die Partitionierung in einem späteren Durchlauf wieder verwenden zu können, wird gespeichert, welche Zellen zu einer Superzelle fusioniert wurden. Damit können die Superzellen später einfach mit den veränderten Parametern neu erstellt werden. Es ist aber auch möglich, eine neue Partitionierung zu erstellen.

3D-Floorplanning und Einfügen der TSVs

Nachdem die Superzellen erstellt wurden, werden diese mit der in Kapitel 4 vorgestellten Datenstruktur Sequenz-Paar und einem der in Kapitel 5 vorgestellten heuristischen Optimierungsverfahren auf mehreren Ebenen kostenoptimal platziert. Ist die Anzahl an Ebenen nicht vorgegeben, so wird für 2, 3, 4, ... Ebenen je eine Platzierung errechnet und die Lösung mit den kleinsten Kosten weiter bearbeitet.
Ist die Platzierung der Superzellen auf k Ebenen gefunden, so sind noch die TSVs einzufügen. Diese sind in die Leitungen zu integrieren, deren Superzellen auf verschiedenen Ebenen liegen. Dabei ist darauf zu achten, die TSVs möglichst zwischen den Superzellen der Leitung zu platzieren, um eine Zunahme der Leitungslänge zu verhindern. In Abbildung 1.7 wurde dieser Bereich mit der Farbe Orange eingezeichnet. Die Superzellen der Leitung sind durch graue Rechtecke dargestellt, die Anschlüsse der Zelle werden in ihrer Mitte angenommen.

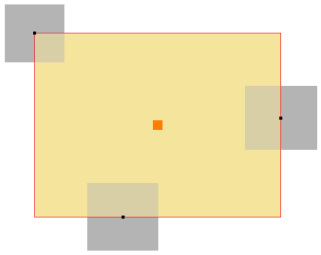

Abb. 1.7: Einfügen der TSVs in eine Leitung

Die TSVs können dabei auch innerhalb der Superzellen platziert werden, da dort einerseits aufgrund des Vergrößerungsfaktors bei der Flächenberechnung der Superzelle ein gewisser Freiraum enthalten ist und andererseits die Zellen der Superzelle beim Platzieren und Verdrahten mit dem kommerziellen P&R-Tool beliebig verschoben werden können. Da unter dem TSV keine Zellen und Leitungen liegen sollten, um eine optimale Ankontaktierung zu gewährleisten, ist dort eine Freifläche einzuplanen. Das Einfügen der TSVs besteht also aus deren Platzierung, dem Reservieren der Freifläche darunter und deren Integration in Leitungen.

Verfeinern

Beim Verfeinern wird für jede Ebene eine Netzliste erzeugt. Dazu werden die Superzellen wieder auf ihre Zellen zurückgeführt und diese erhalten als Vorplatzierung die Mittelpunkt-Koordinaten der Superzelle. Die TSVs und die darunterliegenden reservierten Freiflächen werden fest an den berechneten Positionen platziert. Aus der ursprünglichen Netzliste werden die Leitungen ausgelesen und den Zellen der Ebene und den TSVs zugeordnet. Dadurch entsteht für jede Ebene eine separate Netzliste.

Ebene für Ebene platzieren und verdrahten

Der letzte Schritt eines einzelnen Durchlaufs ist das Platzieren und Verdrahten der einzelnen Ebenen mit dem kommerziellen P&R-Tool Astro. Wie bereits erwähnt, können die Standardzellen dabei beliebig auf der Ebene verschoben werden, die Platzierung der TSVs hingegen ist unveränderbar. Dadurch, dass die TSVs schon fest platziert sind, können die einzelnen Ebenen unabhängig voneinander bearbeitet werden. Sollte die Fläche des Floorplans zu klein bemessen worden sein, so kann diese an jener Stelle auch vergrößert werden.

Parameter anpassen

Nachdem für jede Ebene das Layout mit dem kommerziellen P&R-Tool erstellt worden ist, können die daraus erhaltenen Ergebnisse genutzt werden, um einen neuen Durchlauf mit veränderten Parametern zu starten. Das ist vor allem dann wichtig, wenn das Layout aufgrund der Nichteinhaltung der Zeitbedingung unzulässig ist. Aber auch, wenn die Schätzungen im Floorplan sich stark von den berechneten Werten im Layout unterscheiden, kann ein neuer Durchlauf mit angepassten Parametern das Endergebnis verbessern.

Der Schwerpunkt der Diplomarbeit liegt im 3D-Floorplanning, also in der Platzierung der Superzellen auf mehreren Ebenen. Dazu sollen verschiedene heuristische Optimierungsverfahren zum Einsatz kommen und miteinander verglichen werden. Untersucht werden soll, welches Verfahren am schnellsten gute Resultate liefert und mit welchem in einem gegebenen, längeren Zeitraum das beste Ergebnis ermittelt wird.

2 Modellierung des elektrischen Schaltkreises

Der gegebene elektrische Schaltkreis besteht aus vielen Zellen und Leitungen. Eine Zelle hat je nach Funktion mehrere Anschlüsse, auch Pins genannt, über die das Signal eingespeist bzw. weitergeleitet wird. Dabei werden Ein- und Ausgangspins unterschieden. Die Leitungen dienen der gerichteten Übertragung des Signals zwischen den Zellen. Kein Signal passiert eine Zelle doppelt.

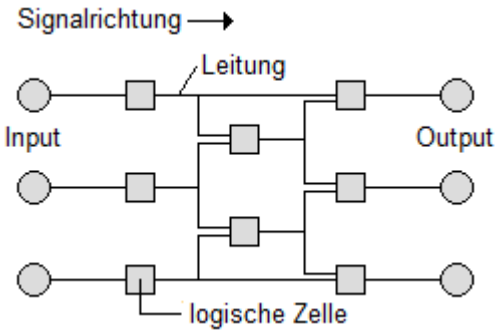

Abb. 2.1: Schematische Darstellung eines elektrischen Schaltkreises

In Abbildung 2.1 werden die IOs durch ihre runde Form gesondert gekennzeichnet. Die übrigen Zellen werden im Folgenden als logische Zellen bezeichnet. Die Signalrichtung soll zeigen, wie das Signal durch die Leitungen übertragen wird. Die Zellen am linken Ende der Leitungen (siehe Abbildung) seien die Zellen, die das Signal in die Leitung einspeisen. Die Zellen am anderen Ende erhalten das Signal und verarbeiten es weiter. Der elektrische Schaltkreis kann sehr gut mit einem Hypergraphen [Ber89] modelliert werden.

Definition 2.1

Ein **Hypergraph** $H = (V, E)$ besteht aus einer Knotenmenge V und einer Menge von Hyperkanten $E = \{e \subseteq V \mid |e| \geq 2\}$.

Definition 2.2

Ein **Kantengewicht** ist eine Abbildung $w : E \to \mathbb{R} : e \mapsto w(e)$, die jeder Kante eine reelle Zahl zuordnet.

Definition 2.3

Die **Nachbarkanten** eines Knotens v sind die Hyperkanten, die diesen enthalten: $E[v] = \{e \in E \mid v \in e\}$. Die Menge der **Nachbarknoten** eines Knotens v ist die Menge $N[v] = \bigcup_{e \in E[v]} e \cap \{v\}$.

Definition 2.4

Ein **Pin** ist der Anschluss einer Kante an einen Knoten, jeder Knoten hat für jede Nachbarkante genau einen Anschluss.
Es sei $N_{Pin} = \sum_{v \in V} |E[v]|$ die Anzahl der Pins des Hypergraphen. Außerdem sei $p_{\max} = \max\{|e| \mid e \in E\}$ die Mächtigkeit der Kante mit den meisten Knoten.

Definition 2.5

Eine Hyperkante e ist **inzident** zu einem Knoten v, falls sie diesen enthält: $v \in e$.
Zwei Knoten u und v heißen **adjazent**, falls es eine Hyperkante gibt, die beide Knoten enthält: $\exists e \in E : \{u, v\} \subseteq e$.

Definition 2.6

Es sei $H_{eS} = (V_{IO} \cup V_L, E)$ der Hypergraph, der den elektrischen Schaltkreis modelliert.
Die Knoten sind durch die Funktion $g : V \to \mathbb{R}^2$ gewichtet.
Die Mengen V_{IO} und V_L bilden zusammen die Knotenmenge $V = V_{IO} \cup V_L$, dadurch werden die Zellen der elektrischen Schaltung repräsentiert, unterschieden nach IOs und logischen Zellen.
Die Leitungen werden dargestellt durch die Menge der Hyperkanten $E = \{e \subseteq V \mid |e| \geq 2\}$.
Die Funktion $g : V \to \mathbb{R}^2 : v \mapsto g(v) = (a, b)$, die jedem Knoten ein Rechteck der Breite a und Höhe b zuordnet, modelliert die Größe der einzelnen Zellen.
Alle Hyperkanten $e \in E$ haben das Kantengewicht $w(e) = 1$.

3 Der Hierarchische Ansatz - Das Erstellen von Superzellen

Der hierarchische Ansatz für die Platzierung der Zellen auf den Ebenen wird genutzt, da die Anzahl an Zellen sehr groß ist. Der Schaltkreis, an dem das Programm getestet wurde, enthält 37 000 Zellen und gehört damit zu den kleinen Beispielen. Ohne eine Verkleinerung des Hypergraphen würde die Suche nach einer guten Platzierung auf den Ebenen mit einem heuristischen Optimierungsverfahren mehrere Wochen in Anspruch nehmen. Deswegen ist es sinnvoll, vorher mehrere Zellen zu einer Superzelle zusammenzufassen. Dieser Prozess wird **Vergröbern** genannt.
Beim Vergröbern ist darauf zu achten, dass möglichst viele Leitungen innerhalb einer Superzelle verlaufen und damit nur wenige Leitungen zwischen diesen entstehen, da die Superzellen immer vollständig auf einer Ebene zu platzieren sind und zwischen den Ebenen möglichst wenige TSVs platziert werden sollen. Für jede Leitung, deren Zellen auf verschiedenen Ebenen liegen, ist später für jede zu passierende Ebene genau ein TSV einzufügen. Es ist dabei unerheblich, wie viele Zellen die Leitung hat und wie diese genau auf den Ebenen verteilt sind. Dieser Sachverhalt wird in Abbildung 3.1 verdeutlicht. Es ist ein Netz mit vier Pins zu sehen, deren Zellen auf zwei Ebenen platziert wurden. Zwischen den beiden Ebene befindet sich immer nur ein TSV für dieses Netz.

Abb. 3.1: Aufteilung eines Netzes auf zwei Ebenen

Je mehr Leitungen zwischen den Superzellen verlaufen, desto mehr TSVs müssen in einem 3D-Aufbau eingefügt werden, um die Verbindung zwischen den Ebenen herzu-

stellen. Deswegen sollte ein guter Algorithmus zum Partitionieren des Hypergraphen in mehrere Teile genutzt werden, um aus jeder Knotenteilmenge der Partition eine Superzelle zu erstellen.

3.1 Das Partitionierungsproblem

Das Partitionierungsproblem umfasst das Finden einer Partition mit minimalem Schnitt:

Definition 3.1
Eine **Partition** P eines Hypergraphen $H = (V, E)$ ist eine disjunkte Zerlegung der Knotenmenge V in l nichtleere Teilmengen:

$$P = \{V_1, V_2, \ldots, V_l\}$$
$$V = \bigcup_{i=1}^{l} V_i$$
$$\forall i, j = 1 \ldots l,\ i \neq j: \quad V_i \cap V_j = \emptyset$$

Die Mengen $V_1, V_2, \ldots, V_l$ der Partition werden im Folgenden als **Block** bezeichnet.

Definition 3.2
Sei $H = (V, E)$ ein Hypergraph mit dem Kantengewicht $w : E \to \mathbb{R} : e \mapsto w(e)$ und der Partition $P = \{V_1, V_2, \ldots, V_l\}$.
Der **Schnitt** $S(P)$ der Partition ist dann gleich der Summe der Gewichte der geschnittenen Kanten E_S – also der Kanten, die Endknoten in verschiedenen Blöcken der Partition haben.

$$E_S = \{e \in E \mid \exists v, u \in e : v \in V_i, u \in V_j, i \neq j\}$$
$$S(P) = \sum_{e \in E_S} w(e)$$

Da die Superzellen in etwa gleich groß sein sollten, muss die Partition ausbalanciert sein. Das heißt, dass alle Blöcke der Partition eine ähnliche Größe haben sollen.

Definition 3.3
Sei $H = (V, E)$ ein Hypergraph, dessen Knoten mit der Funktion $g : V \to \mathbb{R}^2 : v \mapsto g(v) = (a, b)$ gewichtet sind, und $P = \{V_1, V_2, \ldots, V_l\}$ eine Partition des

Hypergraphen. Die Größe eines Knotens ergibt sich aus der Fläche des Rechtecks der zugehörigen Zelle $\tilde{g}(v) = a \cdot b$.
Die **Größe** $\tilde{G}$ der Knotenteilmenge $V_i \in P$ ist die Summe aus den entsprechenden Knotengrößen der Teilmenge: $\tilde{G}(V_i) = \sum_{v \in V_i} \tilde{g}(v)$.

Definition 3.4
Eine Partition ist **ausbalanciert** mit dem Balancefaktor $\alpha \in [0, 1]$, falls für alle Blöcke V_i, $i = 1, 2, \ldots, l$, gilt:

$$\alpha \cdot \frac{\tilde{G}(V)}{l} \leq \tilde{G}(V_i) \leq \frac{1}{\alpha} \cdot \frac{\tilde{G}(V)}{l}$$

Diese Bedingung wird als **Balancebedingung** bezeichnet.

3.2 Bipartitionierung nach Fiduccia-Mattheyses

Der Bipartitionierungsalgorithmus von Fiduccia und Mattheyses [FM82], im Folgenden FM-Algorithmus genannt, wurde wegen seiner geringen Komplexität und der recht guten publizierten Ergebnisse ausgewählt. Allerdings hat sich im Laufe der Arbeit herausgestellt, dass der FM-Algorithmus für große Graphen ungeeignet ist, weswegen bei diesen eine hierarchische Partitionierung, wie in Kapitel 3.3 vorgestellt, auf Grundlage des FM-Algorithmus durchgeführt werden sollte.
Der Algorithmus gehört zu den Bipartitionierungsalgorithmen, das heißt, er unterteilt die Knotenmenge in zwei ausbalancierte Blöcke V_1 und V_2, und ist wie folgt aufgebaut: Als erstes wird eine Startpartition $P^0 = \{V_1^0, V_2^0\}$ erzeugt, alle Knoten als frei markiert und deren Knotenbewertungen berechnet. Die Bewertung eines Knotens gibt an, um welchen Wert sich der Schnitt verkleinert, wenn der Knoten in die andere Knotenteilmenge verschoben wird. Anschließend wird in jedem Schritt der freie Knoten mit der höchsten Wertung verschoben, auch wenn diese negativ ist und damit eine Verschlechterung des Schnittes hervorruft. Soll eine Balancebedingung eingehalten werden, so kann nur dann ein Knoten aus einer beliebigen Knotenteilmenge verschoben werden, wenn diese erfüllt ist. Andernfalls muss ein Knoten aus dem größeren in den kleineren Block verschoben werden, um das Größenverhältnis der Blöcke wieder auszugleichen. Nach dem Verschieben eines Knotens wird dieser als fest markiert

und es werden die Bewertungen der Nachbarknoten aktualisiert. Dies wird solange fortgeführt, bis es keinen freien Knoten mehr gibt, der verschoben werden könnte. Da auch Verschlechterungen akzeptiert werden, ist bei diesem Algorithmus die Partition mit dem besten Schnitt nicht zwangsläufig die letzte erhaltene Partition. Deswegen muss das beste Ergebnis mitgeführt werden. Diese Partition wird am Ende des Algorithmus ausgegeben.

Algorithmus 3.2.1: Bipartitionierung nach Fiduccia und Mattheyses

Eingabe : Hypergraph $H = (V, E)$
Ausgabe : Partition P mit minimalem Schnitt
begin
- $i = 0$
- Erzeuge $P^0 = \{V_1^0, V_2^0\}$
- $\forall v \in V$: $v \leftarrow$ frei
- Berechnung der Knotenbewertung: Algorithmus 3.2.2
- Berechne $S(P^0)$
- **while** *es gibt freie Knoten* **do**
  - **if** *Balancebedingung = true* **then**
    - wähle $u \in \{v \in V \mid v = frei\}$ mit $\max\{b(v)\}$
  - **else**
    - sei $V_j^i \in P^i$, $j = 1, 2$ mit $\max\{\tilde{G}(V_1^i), \tilde{G}(V_2^i)\}$
    - wähle $u \in \{v \in V_j^i \mid v = frei\}$ mit $\max\{b(v)\}$
  - Aktualisieren der Knotenbewertungen: Algorithmus 3.2.3
  - $u \leftarrow$ fest
  - **if** $u \in V_1^i$ **then**
    - $P^{i+1} = \{V_1^{i+1}, V_2^{i+1}\} = \{V_1^i \setminus \{u\}, V_2^i \cup \{u\}\}$
  - **else**
    - $P^{i+1} = \{V_1^{i+1}, V_2^{i+1}\} = \{V_1^i \cup \{u\}, V_2^i \setminus \{u\}\}$
  - $S(P^{i+1}) = S(P^i) - b(u)$
  - $i = i + 1$

Der Algorithmus 3.2.1 stellt die Verfahrensweise der Bipartitionierung nach Fiduccia und Mattheyses als Pseudo-Code dar. Das Erstellen der Startpartition, sowie das Berechnen und Aktualisieren der Knotenbewertungen wird im Folgenden näher beschrieben.

Die Startpartition

Die Startpartition P^0 soll die Balancebedingung erfüllen. Im Allgemeinen wird mit einer zufälligen Halbierung der Knotenmenge gestartet. Günstig ist es, den FM-Algorithmus mehrmals nacheinander auszuführen und dabei mit der Lösung des vorherigen Durchlaufs zu starten.

Berechnung der Knotenbewertung

Die Knotenbewertung eines Knotens v entspricht der Verbesserung des Schnittes durch Verschieben des Knotens in die andere Knotenteilmenge. Zu deren Berechnung wird die Bewertung aller Knoten zunächst auf Null gesetzt. Anschließend werden alle Hyperkanten untersucht und die Knotenbewertungen abhängig von der Aufteilung der Knoten der Hyperkante auf die Blöcke V_1^0 und V_2^0 der Partition P^0 verändert.

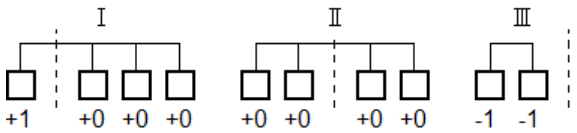

Abb. 3.2: Berechnung der Bewertung der Knoten einer Hyperkante

Wie in Abbildung 3.2 zu sehen, gibt es drei grundlegende Fälle, wie die Knoten einer Kante auf die beiden Knotenteilmengen verteilt sein können. Die unter den Knoten notierten Zahlen +1, +0 und –1 geben an, wie sich die Bewertung des Knotens ändert, wenn diese Kante untersucht wird. Diese drei Fälle werden im Folgenden näher beschrieben:

1. Ein Knoten einer Kante liegt allein in einem der beiden Blöcke, alle anderen Knoten der Kante liegen in der anderen Knotenteilmenge.

Diese Situation wird durch Fall I in Abbildung 3.2 dargestellt. Da die Kante nach dem Verschieben des alleine in einem Block liegenden Knotens nicht mehr geschnitten wäre, steigt dessen Bewertung um Eins.

2. In jeder Partition liegt mehr als ein Knoten der Kante.

Liegt in jeder der beiden Knotenteilmengen mehr als ein Knoten einer Kante, wie in Fall II dargestellt, so bewirkt das Verschieben eines einzelnen Knotens keine Veränderung des Schnittes dieser Kante. Deswegen bleibt die Knotenbewertung unverändert.

3. Die Kante wird nicht geschnitten.

Befinden sich alle Knoten einer Kante in derselben Knotenteilmenge, so ist diese keine Schnittkante (Fall III). Das Verschieben eines der Knoten würde einen neuen Schnitt hervorrufen, weswegen die Bewertung aller Knoten der Kante um Eins sinkt.

Das Berechnen der Knotenbewertung wird in dem Algorithmus 3.2.2 als Pseudo-Code dargelegt.

Algorithmus 3.2.2: Berechnung der Knotenbewertung

Eingabe : Hypergraph $H = (V, E)$ mit Partitionierung $P^0 = \{V_1^0, V_2^0\}$
Ausgabe : Knotenbewertung $b(v)$ $\forall v \in V$
begin
  $\forall v \in V$: $b(v) = 0$
  for $e \in E$ **do**
    $e = e_1 \cup e_2$ mit $e_i = \{u \mid u \in e \wedge u \in V_i^0\}$, $i = 1, 2$
    if $e_1 = \emptyset$ **or** $e_2 = \emptyset$ **then**
      $\forall u \in e$: $b(u) = b(u) - 1$
    else if $|e_1| = 1$ **or** $|e_2| = 1$ **then**
      Sei $\{u\} = e_1$ bzw. $\{u\} = e_2$: $b(u) = b(u) + 1$

Aktualisieren der Knotenbewertung

Nachdem ein Knoten ausgewählt und verschoben wurde, sind die Bewertungen der Nachbarknoten zu aktualisieren.

Bei der Aktualisierung der Knotenbewertung werden alle Nachbarkanten des verschobenen Knotens der Reihe nach untersucht. Die Aktualisierung der Knotenbewertung in jeder Kante erfolgt in zwei Schritten: Zuerst werden die Bewertungen bezüglich dieser Kante auf Null gesetzt, im Anschluss daran wird der zu verschiebende Knoten in der Kante verschoben und die Bewertungen der Knoten neu ermittelt.

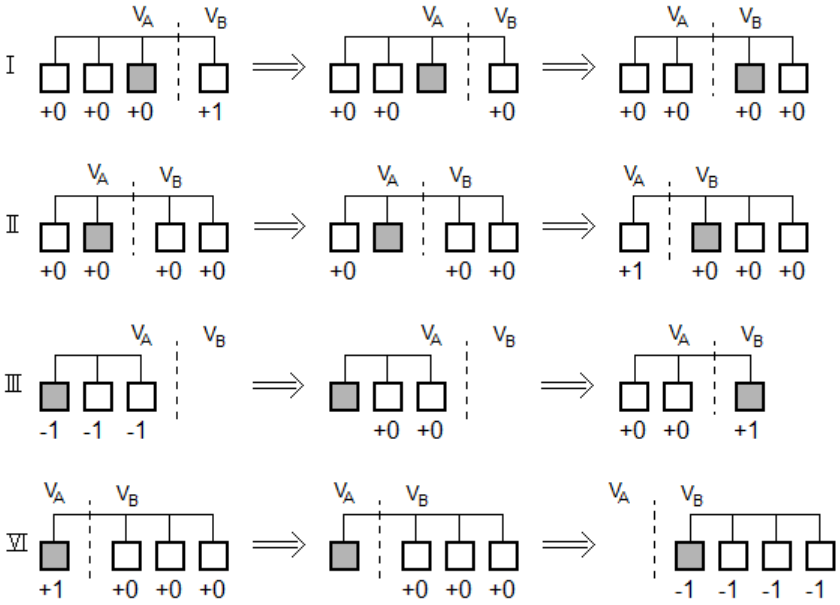

Abb. 3.3: Veränderung der Knotenbewertung durch das Verschieben eines Knotens

In Abbildung 3.3 ist das Verschieben eines Knotens und das Aktualisieren der Bewertungen der Nachbarknoten bezüglich einer Kante beispielhaft an vier Fällen gezeigt.

Auf der linken Seite ist die Ausgangssituation dargestellt, der grau markierte Knoten soll verschoben werden. In der Mitte ist das Ergebnis des ersten Schrittes gezeigt, der zu verschiebende Knoten hat keine Knotenbewertung, alle anderen haben die Wertung Null in dieser Kante. Das Ergebnis des zweiten Aktualisierungsschrittes ist auf der rechten Seite der Abbildung dargestellt, alle Knoten haben die korrekte Bewertung in dieser Kante.

Im Folgenden wird beschrieben, wie die beiden Schritte zur Aktualisierung der Knotenbewertung ablaufen.
Sei u der Knoten, der von der Knotenteilmenge V_A in die Teilmenge V_B verschoben wird.
1. Schritt: Setze die Bewertung der Knoten bezüglich der Kante auf Null.
Lagen vor dem Verschieben von u alle Knoten in V_A, so hatten alle die Bewertung -1 bezüglich dieser Kante (Fall III in Abbildung 3.3), da eine Verschiebung eines jeden Knotens einen Schnitt dieser Kante bewirkt hätte. Um die Bewertung dieser Knoten bezüglich dieser Kante auf Null zu setzen, wird diese um Eins erhöht.
Wenn vor dem Verschieben ein Knoten alleine in V_B lag (Fall I), so bewirkte das Verschieben dieses Knotens eine Verringerung des Schnittes um Eins, was die Bewertung des Knotens um Eins erhöht hat. Das ist nach dem Verschieben von u nicht mehr richtig, dementsprechend wird die Bewertung dieses vorher einzelnen Knotens um Eins gesenkt.
Damit haben nun alle Knoten außer u eine Bewertung von Null in dieser Kante.
2. Schritt: Bewerte die Knoten neu.
Nun wird untersucht, wie das Verschieben der einzelnen Knoten den Schnitt in der neuen Konfiguration beeinflusst:
Liegt nach dem Verschieben von u ein anderer Knoten der Kante allein in einer Partition – diese kann nur V_A sein – wie in Fall II, so bewirkt dessen Verschieben eine Verringerung des Schnittes bezüglich dieser Kante. Damit ist seine Bewertung um Eins zu erhöhen.
Liegen nach dem Verschieben alle Knoten in V_B (Fall IV), so muss die Bewertung aller Knoten um Eins sinken, da das Verschieben eines jeden Knotens einen Schnitt der Kante nach sich ziehen würde.

In dem Aktualisierungsschritt werden alle Nachbarkanten von u untersucht und die Knotenbewertung wie eben beschrieben angepasst. Die Bewertung von u wird auf den negierten Wert der Bewertung vor dem Verschieben gesetzt, dies ist aber nur dann relevant, wenn Knoten mehrfach verschoben werden dürfen, wie auf Seite 22 erläutert.

Algorithmus 3.2.3: Aktualisieren der Knotenbewertungen

Eingabe : u ... Knoten, der verschoben wurde
A ... Knotenteilmenge, in der u vor dem Verschieben lag
B ... Knotenteilmenge, in der u nach dem Verschieben liegen wird
Ausgabe : Knotenbewertung $b(v)\ \forall v \in N[u]$
begin
  $b(u) \leftarrow -b(u)$
  for $e \in N[u]$ **do**
    $e = e_A \cup e_B$ mit $e_I = \{v \in e \mid v \in I\}$, $I = A, B$
    if $e_B = \emptyset$ **then**
      $\forall v \in e$: $b(v) \leftarrow b(v) + 1$
    else if $|e_B| = 1$ **then**
      Sei $\{v\} = e_B$: $b(v) \leftarrow b(v) - 1$
    $e_A \leftarrow e_A \setminus \{u\}$
    $e_B \leftarrow e_B \cup \{u\}$
    if $e_A = \emptyset$ **then**
      $\forall v \in e$: $b(v) \leftarrow b(v) - 1$
    else if $|e_A| = 1$ **then**
      Sei $\{v\} = e_A$: $b(v) \leftarrow b(v) + 1$

Komplexitätsbetrachtungen:

Wie bereits aus Kapitel 2 bekannt, ist $N_{Pin} = \sum_{v \in V} |E[v]|$ die Anzahl der Pins und $p_{\max} = \max\{|e| \mid e \in E\}$ die Mächtigkeit der Kante mit den meisten Knoten.

Satz 3.1. *Findet das Suchen des Knotens mit maximaler Bewertung in linearer Zeit statt, so hat der Bipartitionierungsalgorithmus nach Fiduccia und Mattheyses die Komplexität* $\mathcal{O}(p_{max} \cdot N_{Pin})$.

Beweis:

1. Die Berechnung der Knotenbewertungen ist aus $\mathcal{O}(N_{Pin})$:
   Zur Berechnung der Knotenbewertung zu Beginn des Algorithmus muss jede der $|E|$ Kanten untersucht werden. Für jede Kante $e \in E$ müssen dabei höchstens $|e|$ Knotenbewertungen verändert werden. Das sind $\sum_{e \in E} |e|$ Schritte.
   Zu zeigen ist noch, dass gilt: $\sum_{e \in E} |e| = \sum_{v \in V} |E[v]| = N_{Pin}$.

Würde die Gleichheit nicht gelten, so gäbe es entweder eine Kante mit einem Endknoten, der nicht in V liegt oder einen Knoten mit einer inzidenten Kante, die nicht in E liegt. Diese beiden Fälle sind aber aufgrund der Definition einer Hyperkante und der Menge der Nachbarkanten aus Kapitel 2.2 ausgeschlossen.

2. Das Verschieben eines Knotens $v \in V$ ist aus $\mathcal{O}(\sum_{e \in E[v]} |e|)$:
   Das Verschieben eines Knotens besteht zum einen aus dem Umsetzen des Knotens in die andere Knotenteilmenge und zum anderen aus der Aktualisierung der Knotenbewertungen der Nachbarknoten. Um die Knotenbewertungen zu aktualisieren, werden $|E[v]|$ Kanten betrachtet. Für jede Kante $e \in E[v]$ müssen höchstens $|e|$ Knotenbewertungen verändert werden. Die Komplexität beträgt damit $\mathcal{O}(\sum_{e \in E[v]} |e|)$.

3. Das einmalige Verschieben aller Knoten ist aus $\mathcal{O}(p_{\max} \cdot N_{Pin})$:
   Im schlechtesten Fall wird jeder Knoten genau einmal verschoben. Damit beträgt der Aufwand für die Gesamtheit aller Verschiebungen: $\sum_{v \in V} \sum_{e \in E[v]} |e|$. Jede Kante $e \in E$ wird von jedem inzidenten Knoten einmal in die Summe eingebracht, also $|e|$-mal. Insgesamt ergibt sich damit ein Aufwand von $\sum_{e \in E} |e|^2 \leq p_{\max} \cdot \sum_{e \in E} |e| = p_{\max} \cdot N_{Pin}$.

Da $p_{\max}$ im Allgemeinen recht klein ist – die Leitungen eines elektrischen Schaltkreises haben meist keine zehn Pins – kann für die meisten Hypergraphen eine Komplexität von $\mathcal{O}(N_{Pin})$ angenommen werden.
Allerdings wurde der Algorithmus hier nicht so implementiert, dass die Suche nach dem Knoten mit der besten Bewertung in linearer Zeit stattfindet, sondern in $\mathcal{O}(|V|)$.

Ein Nachteil des Algorithmus ist, dass jeder Knoten höchstens einmal verschoben werden darf und damit eine einmal vollzogene Verschiebung nicht rückgängig gemacht werden kann, auch wenn dies in einem späteren Iterationsschritt vernünftig wäre.
Dürfte nacheinander derselbe Knoten mehrmals verschoben werden, so würden Verschlechterungen fast immer sofort rückgängig gemacht, da die negative Bewertung des Knotens durch das Verschieben negiert wird und damit eine positive Bewertung entsteht. Häufig wird dieser Knoten dann die beste Bewertung haben und wieder zurückgeschoben werden. Das verhindert ein Entkommen aus lokalen Minima. Es kann auch passieren, dass immer wieder derselbe Knoten hin und her geschoben

wird. Um derartige Probleme zu umgehen und trotzdem ein wiederholtes Verschieben eines Knotens zu erlauben, wurde in dieser Arbeit eine Art **Warteschleife** (**WS**) eingerichtet. Nach dem Verschieben eines Knotens ist er für eine vorgegebene Anzahl an Schritten gesperrt und kann erst danach wieder verschoben werden. Diese Schrittanzahl entspricht der Länge der Warteschlange. Insgesamt darf jeder Knoten höchstens W-mal verschoben werden. Es hat sich gezeigt, dass sich die Ergebnisse damit verbessern, allerdings nicht in dem selben Maße, wie durch das wiederholte Ausführen des Algorithmus mit der Lösung des vorherigen Durchlaufs.

Im Folgenden werden die Ergebnisse, die mit den verschiedensten Varianten des FM-Algorithmus erzielt wurden, aufgezeigt.
Es wurde für jeden Versuch ein neuer Zufallsgraph mit der entsprechenden Pinanzahl erzeugt und anschließend mit jeder Methode einmal mit dem Balancefaktor $\alpha = 0.9$ halbiert. Der Graph ist ähnlich wie der Hypergraph der hier verwendeten Beispiel-Schaltung aufgebaut: Jeder Knoten hat im Schnitt 3.5 Pins und jede Kante besitzt durchschnittlich 3 Pins. Außerdem besitzt der Graph eine gewisse Linearität, damit ist gemeint, dass die Knoten der Hyperkanten nicht beliebig aus allen Knoten ausgewählt werden dürfen, sondern nur aus Knoten eines bestimmten Bereiches. Um dies nachzubilden, wurden die Knoten des Hypergraphen in eine numerische Ordnung gebracht und die Kanten zwischen Knoten erzeugt, die innerhalb eines Intervalls liegen. Die Länge des Intervalls wurde auf ein Prozent der Knotenanzahl gesetzt.

Pinanzahl / Methode	500		1 000		5 000	
	Schnitt	Zeit	Schnitt	Zeit	Schnitt	Zeit
1x FM	19.36	0.01	33.62	0.01	287.17	0.59
3x FM	9.75	0.01	18.06	0.02	102.61	0.68
5x FM	7.92	0.01	14.96	0.02	70.94	0.78
10x FM	6.51	0.01	13.75	0.04	62.56	1.03
1x FM mit WS1, $W = 2$	16.69	0.00	26.24	0.00	134.03	0.10
5x FM mit WS1, $W = 2$	8.30	0.01	13.29	0.02	72.51	0.30
1x FM mit WS1, $W = 5$	14.07	0.00	23.64	0.01	122.51	0.25
2x FM mit WS1, $W = 5$	11.38	0.01	19.26	0.02	99.70	0.30
1x FM mit WS1, $W = 10$	14.75	0.01	24.02	0.03	114.72	0.50
1x FM mit WS2, $W = 2$	12.75	0.00	25.45	0.00	118.79	0.10
5x FM mit WS2, $W = 2$	7.52	0.01	14.34	0.02	68.25	0.30
1x FM mit WS2, $W = 5$	10.93	0.00	21.23	0.01	111.21	0.24
2x FM mit WS2, $W = 5$	8.82	0.00	17.89	0.02	97.19	0.29
1x FM mit WS2, $W = 10$	10.86	0.01	20.85	0.02	105.17	0.47

Tab. 3.1: Ergebnisse für den FM-Algorithmus gemittelt aus 100 Versuchen, Graph mit durchschnittlich 3 Pins pro Kante und 3.5 Pins pro Knoten

In Tabelle 3.1 ist zu sehen, wie die verschiedenen Varianten des FM-Algorithmus abschneiden. Die Algorithmen WS1 und WS2 sind FM-Variationen mit Warteschlange. Die Länge der Warteschlange beträgt bei WS1 1% der Knotenanzahl und bei WS2 10% der Knotenanzahl. Dabei gibt W an, wie oft ein Knoten höchstens verschoben werden kann.
Bei dem mehrmaligen Ausführen des FM-Algorithmus (3x FM, 5x FM und 10x FM) wurde immer mit der besten je erhaltenen Lösung gestartet.
An den Ergebnissen ist zu sehen, dass es beim einfachen FM-Algorithmus am besten ist, diesen mehrmals zu wiederholen. Die Warteschlangen dagegen bringen kaum Vorteile. Auch hier werden durch mehrmaliges Ausführen bessere Ergebnisse erzielt. So schneidet 5x FM mit WS und W=2 deutlich besser ab, als 1x FM mit WS und W=10, obwohl die gleiche Zahl an Iterationen ausführbar war.
Allerdings zeigte der Vergleich mit der hierarchischen Partitionierung, dass der FM-Algorithmus schon bei vergleichsweise kleinen Graphen mit nur 1000 Pins deutlich schlechtere Ergebnisse liefert.

3.3 Hierarchische Partitionierung

Da die Ergebnisse des FM-Algorithmus nicht die Erwartungen erfüllten (bei der Platzierung auf zwei Ebenen waren weit über 4000 TSVs nötig, wodurch sich die nötige Fläche des Chips mehr als verdoppelt hat), wurde eine hierarchische Partitionierung implementiert. Es hat sich gezeigt, dass damit deutlich bessere Ergebnisse erzielt werden.
Hierarchischer Ansatz bedeutet, dass der Graph zunächst verkleinert wird, indem immer wieder ausgewählte Knoten zu einem neuen Knoten zusammengefasst werden, bis eine gewünschte Knotenanzahl erreicht ist. Dann kann auf diesem kleinen Graphen eine FM-Bipartitionierung durchgeführt werden. Im Anschluss daran muss die so erhaltene Lösung auf den Originalgraphen übertragen werden.
Einer der bekanntesten hierarchischen Partitionierungsalgorithmen ist der von G. Karypis et al. in [KAKS99] vorgestellte Algorithmus hMETIS. Dort werden zunächst drei Methoden zum Verkleinern des Hypergraphen vorgestellt:

1. Kantenfusion
2. Hyperkantenfusion

3. Modifizierte Hyperkantenfusion

Ausgehend von dem Originalgraphen H^0 wird durch eine dieser Methoden eine Folge von Graphen $H^i = (V^i, E^i)$, $i = 1, 2 \dots n$ erstellt, deren Knoten- und Kantenanzahl zunehmend kleiner wird. Die Knoten des Graphen H^i wurden durch eine Fusion der Knoten aus H^{i-1} erzeugt. Jedem Knoten $v \in V^i$ wird eine Menge $U(v) \subset V^{i-1}$ von Knoten zugeordnet, aus deren Verschmelzung v entstanden ist. Die Knoten aus U werden im Folgenden als Kinder von v bezeichnet und v als deren Vater. Jedem Knoten aus H^{i-1} wird genau ein Vater zugeordnet, es können also keine zwei Knoten aus H^i den selben Knoten als Kind haben. Außerdem muss jeder Knoten aus H^{i-1} irgend einen Knoten aus H^i als Vater haben, sonst bliebe dieser unberücksichtigt. Es gibt also eine Abbildung $U : V^i \to \mathcal{P}(V^{i-1}) : v \mapsto U(v) = \{u \in V^{i-1} \mid u \text{ ist Kind von } v\}$ von Knoten des Hypergraphen H^i auf die Knoten des Hypergraphen H^{i-1}, so dass $\bigcup_{v \in V^i} U(v) = V^{i-1}$, $\forall v \in V^i : U(v) \neq \emptyset$ und $\forall u, v \in V^i,\ u \neq v : U(v) \cap U(u) = \emptyset$ gilt. Zur Erstellung von H^i aus H^{i-1} soll die Knotenmenge V^{i-1} in disjunkte Teilmengen zerlegt werden, um aus jeder dieser Teilmengen einen neuen Knoten für H^i zu erstellen. Diese Teilmengen werden im Weiteren als **Superknoten** bezeichnet.

Im Folgenden wird beschrieben, wie die drei bereits genannten Methoden bei der Erstellung der Superknoten vorgehen.
Bei der **Kantenfusion** werden ausschließlich Knotenpaare verschmolzen, deswegen enthalten die Superknoten höchstens zwei Knoten. Um diese zu erstellen, werden die Knoten in einer beliebigen Reihenfolge untersucht.
Sei v der Knoten, der untersucht wird. Ist dieser schon einem Superknoten zugeordnet, so wird der nächste Knoten betrachtet. Andernfalls wird der Nachbarknoten von v ermittelt, der am stärksten mit ihm verbunden ist und noch zu keinem Superknoten gehört. Sei $w : E \to \mathbb{R} : e \mapsto w(e)$ das Kantengewicht auf dem Graphen H^{i-1}. Dann wird v mit dem Nachbarknoten $u \in \{z | z \in N[v] \wedge z \notin \mathit{Superknoten}\}$ zu einem neuen Superknoten zusammengefasst, mit dem er das größte Knotenpaargewicht $\tilde{w}(\{u, v\}) = \sum_{e \in N[v] \cap N[u]} w(e)$ hat. Gibt es keinen solchen Nachbarknoten, wenn also alle Nachbarknoten schon einem Superknoten angehören, so bildet v alleine einen neuen Superknoten.
Wenn alle Knoten aus H^{i-1} untersucht worden sind, wird aus jedem der erstellten Superknoten ein neuer Knoten gebildet. Mit dieser Knotenmenge wird dann der

neue Graph H^i erstellt. Die Hyperkanten von H^i ergeben sich aus den Hyperkanten des Graphen H^{i-1}, indem darin alle Knoten durch die entsprechenden Väter ersetzt werden und im Anschluss doppelte Endknoten entfernt werden. Da eine Hyperkante wenigstens zwei Knoten enthalten muss, entfallen alle Hyperkanten mit nur einem Knoten.

Der Nachteil dieser Methode ist, dass die Kantenanzahl nur sehr langsam sinkt. In jedem Schritt fallen nur die Kanten der Mächtigkeit zwei in einem Knoten zusammen, alle anderen Kanten bleiben, wenn auch verkleinert, bestehen. Deswegen wurde die Überlegung angestellt, alle Knoten einer Hyperkante zu verschmelzen. Diese Herangehensweise heißt **Hyperkantenfusion**. Dabei werden die Kanten zunächst in absteigender Ordnung nach ihrem Gewicht und die Kanten eines Gewichts aufsteigend nach der Knotenanzahl geordnet. Anschließend werden die Hyperkanten in dieser Reihenfolge untersucht: In dem Fall, dass alle Knoten der Hyperkante zu keinem Superknoten gehören, bilden diese einen neuen Superknoten. Andernfalls wird die nächste Hyperkante untersucht. Nachdem alle Hyperkanten aus H^{i-1} untersucht worden sind, bildet jeder Knoten, der noch keinem Superknoten angehört, einen eigenen Superknoten, danach wird wiederum aus den Superknoten der Graph H^i erstellt.

Mit dieser Herangehensweise sinkt die Zahl an Hyperkanten deutlich schneller, allerdings entstehen dabei Knoten sehr unterschiedlicher Größe[4], da die Restknoten zu kleinen Superknoten werden. Deswegen haben die Entwickler des Verfahrens hMETIS die Hyperkantenfusion zur **modifizierten Hyperkantenfusion** weiterentwickelt. Bei dieser Strategie wird im ersten Schritt eine Hyperkantenfusion ausgeführt, allerdings wird nach der ersten Untersuchung der Kanten direkt eine zweite angeschlossen: Die Hyperkanten werden dabei wiederum in der zuvor erstellten Reihenfolge untersucht und alle Knoten einer Hyperkante, die noch keinem Superknoten angehören, werden zu einem neuen Superknoten zusammengefasst. Damit sinkt die Anzahl der Kanten und Knoten schnell.

Die Abbildung 3.4 zeigt die drei in [KAKS99] vorgestellten Methoden zur Graphen-Verkleinerung. Die schwarzen und grauen Punkte repräsentieren die Knoten. Die umliegenden Kreise stellen die Hyperkanten dar: Alle Knoten innerhalb eines Kreises gehören zu derselben Hyperkante. Knoten, die von mehreren Kreisen umgeben sind,

[4] Die Größe eines Knotens ergibt sich aus der Summe der Größe der Kinder. In H^0 ist die Größe eines Knotens durch das Knotengewicht $\tilde{g}$ (Vgl. Definition 3.3 auf Seite 14) festgelegt

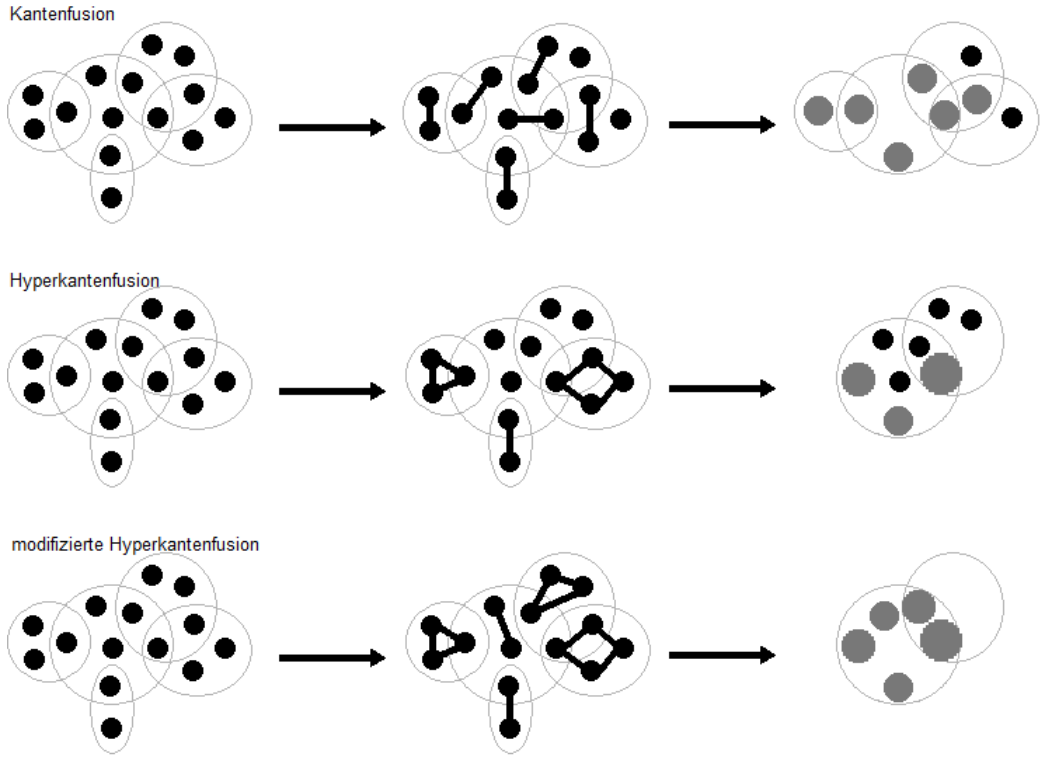

Abb. 3.4: Schematische Darstellungen der Methoden der Graphen-Verkleinerung

gehören mehreren Hyperkanten an.
Das linke Schema des Hypergraphen zeigt den Ausgangszustand, in der mittleren Abbildung wurden jeweils die Knoten durch Linien verbunden, die miteinander fusioniert werden sollen. Ganz rechts ist dann der verkleinerte Hypergraph zu sehen, die neu entstanden Knoten wurden darin grau eingefärbt.
Aufgrund der Vorzüge der modifizierten Hyperkantenfusion wurde diese für den Partitionierungsprozess gewählt. Damit entsteht schnell ein verkleinerter Graph mit recht gleichmäßig großen Knoten. Leider sinkt die Anzahl der Hyperkanten nicht in dem selben Maße wie die der Knoten, so dass dieses Verfahren nicht ausreicht, um den Graphen zu partitionieren und die Superzellen zu erstellen.

Es wird also mit der modifizierten Kantenfusion eine Reihe von verkleinerten Graphen H^i, $i = 0, 1, \ldots n$ erstellt. Auf dem letzten, kleinsten Graphen H^n wird mit dem FM-Bipartitionierer eine Partition $P^n = \{V_1^n, V_1^n\}$ erzeugt.
Mit der Abbildung $U : V^i \to \mathcal{P}(V^{i-1}) : v \mapsto U(v) = \{u \in V^{i-1} \mid u \text{ ist Kind von } v\}$ kann dann aus der Partition $P^i = \{V_1^i, V_2^i\}$ des Graphen H^i die Partition des darunter liegenden Graphen H^{i-1} erzeugt werden:

$$P^{i-1} = \{V_1^{i-1}, V_2^{i-1}\}$$
$$\text{mit } V_1^{i-1} = \bigcup^{v \in V_1^i} U(v) \text{ und } V_2^{i-1} = \bigcup^{v \in V_2^i} U(v).$$

Dieses Zurückgehen von dem Graphen H^n auf den Originalgraphen H^0 wird als **Verfeinern** bezeichnet.

Die bei der Verfeinerung erhaltene Partition könnte aber noch verbessert werden, indem zum Beispiel eine FM-Partitionierung mit P^{i-1} als Startpartition durchgeführt wird. Die Entwickler von hMETIS haben eine beschleunigte Nachverbesserung entwickelt, die **Hyperkanten-Nachbesserung**. Dabei werden nicht nur einzelne Knoten verschoben, sondern ganze Hyperkanten.

Algorithmus 3.3.1: nachträgliche Verbesserung der Knotenpartition mit der Hyperkanten-Nachbesserung

Eingabe : Hypergraph $H^i = (V^i, E^i)$
Knotenpartition $P^i = \{V_1^i, V_2^i\}$
Ausgabe : Knotenpartition $P^i = \{V_1^i, V_2^i\}$
begin
  for $e \in E^i$ **do**
    $e_1 = \{v \in e \mid v \in V_1^i\}$
    $e_2 = \{v \in e \mid v \in V_2^i\}$
    $S = S(P^i)$
    $P^i = \{V_1^i \cup e_2, V_2^i \setminus e_2\}$
    if $S(P^i) > S$ **or** *Balancebedingung = false* **then**
      $P^i = \{V_1^i \setminus e_2, V^{i,2} \cup e_2\}$
    $S = S(P^i)$
    $P^i = \{V_1^i \setminus e_1, V_2^i \cup e_1\}$
    if $S(P^i) > S$ **or** *Balancebedingung = false* **then**
      $P^i = \{V_1^i \cup e_1, V_2^i \setminus e_1\}$

Die Hyperkanten-Nachbesserung geht wie folgt vor:
Nachdem die Partitionierung auf dem Hypergraphen $H^i = \{V^i, E^i\}$ aus der Partitionierung des Hypergraphen H^{i+1} wie oben beschrieben ermittelt wurde, werden die Kanten in einer zufälligen Reihenfolge untersucht und die Endknoten je einer Knotenteilmenge verschoben. Verschlechtert sich der Schnitt dadurch oder wird die Balancebedingung verletzt, so wird die Verschiebung rückgängig gemacht.

In der folgenden Tabelle sollen die Ergebnisse der verschiedenen Varianten des hMETIS-Algorithmus verglichen werden. Es werden zwei Knotengrenzen für das Verkleinern des Graphen und fünf verschiedenen Methoden zur nachträglichen Verbesserung des Schnittes betrachtet. Dabei ist p_G der Prozentsatz an Knoten, auf den der

Graph verkleinert wird und n die durchschnittliche Anzahl an Vergröberungsschritten, die vorgenommen wurde. Die Methoden zur Nachbesserung werden in der Tabelle wie folgt bezeichnet:

- ohne NB: keine Nachbesserung
- mit FM: Nachbesserung mit dem FM-Algorithmus
- mit HN: Hyperkanten-Nachbesserung
- mit FM & HN: abwechselnde Ausführung des FM-Algorithmus und der Hyperkanten-Nachbesserung
- mit 3x FM: dreimaliges Ausführen des FM-Algorithmus zur Nachbesserung

Der FM-Algorithmus auf dem kleinsten Graphen H^n wurde 10-mal ausgeführt und dabei immer mit der besten Lösung der vorherigen Ausführung begonnen. Bei allen drei Methoden erfolgt die Nachbesserung in jedem Verfeinerungsschritt – also immer nachdem die Partition des Graphen H^i auf den Graphen H^{i-1} übertragen wurde – und wird solange wiederholt, bis keine Verbesserung mehr stattfindet.
Die verwendeten Graphen sind dieselben, die auch bei dem FM-Algorithmus zum Einsatz kamen.

Methode \ Pinanzahl	5 000		10 000		20 000	
	Schnitt	Zeit	Schnitt	Zeit	Schnitt	Zeit
1x FM10	58.60	0.47	125.75	1.77	261.61	7.18
n	10.22		10.95		12.22	
hMETIS ohne NB mit $p_G = 5\%$	44.02	0.12	120.82	0.38	278.75	1.37
hMETIS mit FM mit $p_G = 5\%$	24.63	0.49	49.73	1.85	89.56	7.71
hMETIS mit HN mit $p_G = 5\%$	31.89	0.83	69.96	4.26	133.78	21.84
hMETIS mit FM & HN mit $p_G = 5\%$	25.05	1.06	48.75	4.87	89.48	23.11
hMETIS mit 3x FM mit $p_G = 5\%$	20.75	0.87	43.95	3.22	85.03	12.67
n	3		3		3	
hMETIS ohne NB mit $p_G = 20\%$	64.91	0.23	125.83	0.73	231.74	2.64
hMETIS mit FM mit $p_G = 20\%$	42.58	0.42	81.04	1.55	148.36	6.19
hMETIS mit HN mit $p_G = 20\%$	50.15	0.66	95.37	2.63	169.15	10.38
hMETIS mit FM & HN mit $p_G = 20\%$	40.98	0.88	81.79	3.43	150.59	14.19
hMETIS mit 3x FM mit $p_G = 20\%$	36.39	0.74	74.29	2.72	129.07	10.68

Tab. 3.2: Ergebnisse für den hMETIS-Algorithmus,
Graph mit durchschnittlich 3 Pins pro Kante und 3.5 Pins pro Knoten

Wie der Tabelle zu entnehmen ist, werden mit dem hierarchischen Algorithmus hMETIS deutlich bessere Ergebnisse erzielt als mit dem einfachen FM-Algorithmus.

Wird hMETIS ohne Nachbesserung ausgeführt, werden geringfügig bessere Ergebnisse in kürzerer Zeit ermittelt. Bei den Varianten mit Nachbesserung ist aber keine Beschleunigung erkennbar, im Gegenteil, die Laufzeit steigt deutlich an. Allerdings sind die Ergebnisse auch deutlich verbessert worden. Die von den Entwicklern des Algorithmus hMETIS vorgeschlagene Hyperkanten-Nachbesserung hat bei diesen Graphen am schlechtesten abgeschnitten. Die besten Ergebnisse wurden mit der Nachbesserung 3x FM errechnet, da dort mehr Verbesserungsanläufe möglich sind. Bei diesem Verfahren wird erst nach dreimaligem Ausführen des FM-Algorithmus' überprüft, ob eine Verbesserung erfolgte und gegebenenfalls abgebrochen.

3.4 Erstellen der Superzellen

Das Erstellen der Superzellen erfolgt in mehreren Schritten. Aus dem elektrischen Schaltkreis wurde der Hypergraph $H_{eS} = (V_{IO} \cup V_L, E)$ abgeleitet. Die Mengen V_{IO} und V_L bilden die Knoten des Graphen – unterteilt in IOs und in logische Zellen. Die Knoten sind mit der Funktion $g : V \to \mathbb{R}^2$ gewichtet.
Im Layout müssen die IOs am Rand des Chips platziert werden. Deswegen dürfen sie nicht in den Superzellen enthalten sein und werden vor der Partitionierung aus H_{eS} entfernt. Die Kanten zwischen IOs und logischen Zellen werden vollständig gelöscht, da sie in dem verkleinerten Graphen auf jeden Fall enthalten sind und damit in der Partitionierung nicht mehr berücksichtigt werden müssen. Damit entsteht der Hypergraph $H^0_{eS} = (V_L, E^0)$ mit $E^0 = \{e \in E \mid e \subset V_L\}$. Auf H^0_{eS} findet dann die Partitionierung statt.

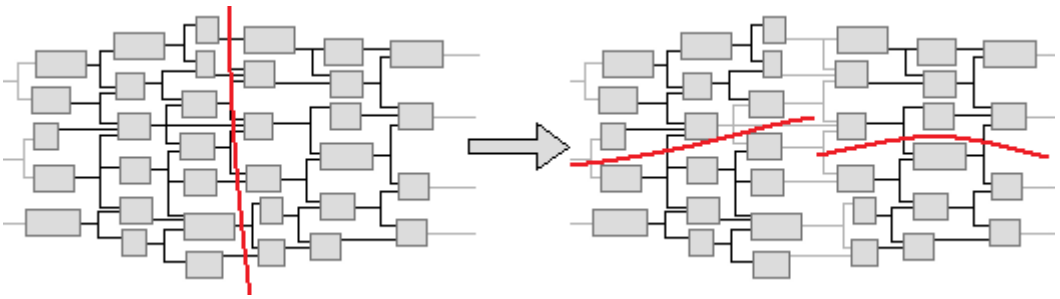

Abb. 3.5: Partitionieren des Hypergraphen H^0_{eS} in vier Blöcke

Abbildung 3.5 zeigt die nun folgende Partitionierung auf H^0_{eS}: Zunächst wird der Hypergraph ohne IOs halbiert (rote Linie in der ersten Grafik von Abbildung 3.5).

Diese Halbierung erfolgt mit dem hier vorgestellten hMETIS-Algorithmus. Dazu wird der Graph mit der modifizierten Hyperkantenfusion verkleinert. Auf dem kleinen Graph wird mit Hilfe des FM-Algorithmus eine ausbalancierte Partition in zwei Knotenteilmengen ermittelt und danach diese auf den Originalgraphen übertragen. Dabei wird schrittweise mit 3x FM verbessert.
Im Anschluss daran wird aus jedem der beiden Blöcke der Partition $P = \{V_1, V_2\}$ des Graphen H_{eS}^0 ein neuer Hypergraph erstellt. Aus den beiden Knotenteilmengen der Partitionierung entstehen die Knotenmengen der neuen Hypergraphen. Als Kanten werden die Kanten des Graphen H_{eS}^0 eingefügt, deren Endknoten alle in der Knotenteilmenge des entstehenden Graphen liegen. Damit entstehen die beiden Graphen $H_{eS}^1 = (V^1, E^1)$ mit $V^1 = V_1 \in P$ und $E^1 = \{e \in E^0 \mid e \subset V^1\}$, sowie $H_{eS}^2 = (V^2, E^2)$ mit $V^2 = V_2 \in P$ und $E^2 = \{e \in E^0 \mid e \subset V^2\}$. Es bleiben im Folgenden alle Kanten, die zwischen den beiden Graphen verlaufen, unberücksichtigt, da sie schon durch den ersten Schnitt entstanden sind. In Abbildung 3.5 sind diese entfallenden Kanten hellgrau eingefärbt.
Anschließend werden die beiden Teilgraphen wieder halbiert. Aus der so entstandenen Partitionierung auf den beiden Teilgraphen werden wieder neue Teilgraphen nach dem oben beschriebenen Schema erstellt. Dieses Vorgehen wird solange fortgeführt, bis die gewünschte Superzellen-Anzahl, eine Potenz von Zwei, entstanden ist. Dann werden die Zellen der Knoten eines jeden Teilgraphen zu einer Superzelle verschmolzen. Die Größe der Superzelle berechnet sich aus der Größe der Zellen, die zu dieser verschmolzen werden, multipliziert mit dem Vergrößerungsfaktor[5]. Gibt es innerhalb einer Superzelle sehr viele Leitungen, so muss gegebenenfalls deren Fläche vergrößert werden, um die Verdrahtung der Zellen untereinander auf dem Chip garantieren zu können. Die Menge der Superzellen bildet aber nicht allein die Knotenmenge, es werden auch die IOs wieder zu dem Graphen hinzugefügt. Diese müssen bei der Platzierung gesondert betrachtet werden und sind deswegen in einer separaten Menge zu speichern.
Wurden die Superzellen erstellt, müssen noch die Hyperkanten eingefügt werden. Dazu werden die Hyperkanten des Originalgraphen untersucht. Die Knoten der Hyperkanten werden durch die entsprechenden Superzellen ersetzt. Im Anschluss werden doppelte Knoten in den Hyperkanten entfernt. Hyperkanten mit nur einem

[5] anzupassender Parameter, siehe Kapitel 1.3

Knoten sind zu löschen.

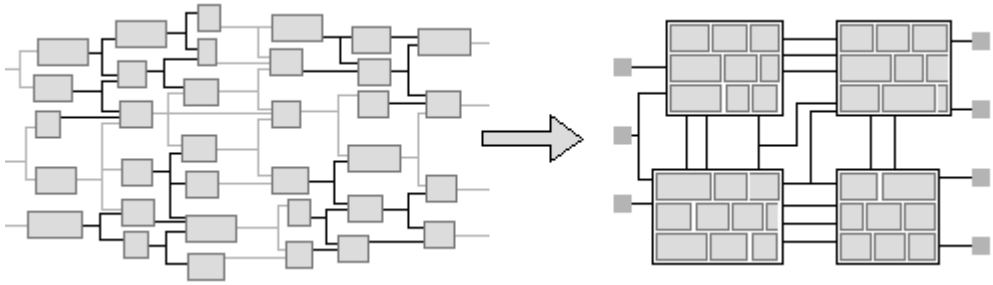

Abb. 3.6: Erstellen der Superzellen aus den Teilgraphen

In Abbildung 3.6 wurde auf der linken Seite die Partitionierung von H^0_{eS} in vier Teile dargestellt und auf der rechten Seite der daraus entstehende Hypergraph für die Platzierung $H_P = (V_{IO} \cup V_S, E_P)$ mit der Menge der Ein- und Ausgänge V_{IO}, der Menge der Superzellen V_S, die zusammen die Knotenmenge des Graphen bilden und der Menge der Hyperkanten E_P. Die Knoten sind wiederum durch die Funktion $g : V_P \to \mathbb{R}^2 : v \mapsto g(v) = (a, b)$ gewichtet.
Es entsteht ein meist sehr dichter Hypergraph mit der gewünschten Knotenanzahl – damit kann schnell und effektiv die Platzierung vorgenommen werden.

4 Datenstrukturen für das Platzieren

Im ersten Abschnitt sollen zunächst einige wichtige Begriffe erläutert werden. Im Anschluss daran werden mehrere Datenstrukturen für eine zweidimensionale Platzierung von rechteckigen Objekten kurz vorgestellt. Die in dem Programm genutzte Datenstruktur Sequenz-Paar wird danach näher erläutert und die Eignung als Datenstruktur für die Berechnung des Floorplans mit einem iterativen Suchverfahren dargelegt. Zum Abschluss wird ein kleiner Einblick in Datenstrukturen zur dreidimensionalen Platzierung gegeben.

4.1 Definitionen

Beim Platzieren sind N Modulen[6] gegebener Größe eine Position in der Ebene zuzuordnen, so dass sich keine zwei Module überschneiden. Eine solche Platzierung wird auch als Floorplan bezeichnet.

Definition 4.1
Die **Fläche einer Platzierung** ist die Fläche des kleinsten Rechtecks, das alle positionierten Module in der Ebene umschließt.

Platzierungen lassen sich in verschiedene Kategorien unterteilen, manche Datenstrukturen können nur eine bestimmte Art von Platzierungen darstellen.

Definition 4.2
Kann die Fläche einer Platzierung solange zwischen den Modulen halbiert werden, bis jedes Modul einzeln vorliegt, wird diese Platzierung **slicing-Floorplan** genannt. Andernfalls liegt ein **nonslicing-Floorplan** vor.

[6] Ein Modul ist ein zu platzierendes Objekt, zum Beispiel eine Superzelle. Im Folgenden seinen die Module immer rechteckig.

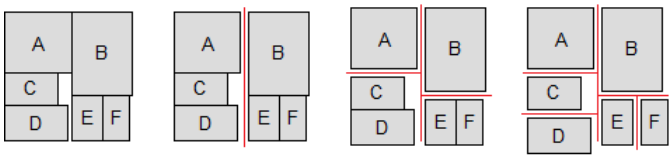

Abb. 4.1: Beispiel für einen slicing-Floorplan

Definition 4.3

Platzierungen, bei denen die Verschiebung eines Moduls nach links oder unten immer eine Verschiebung anderer Module zur Folge hat, werden als **kompaktierte Platzierungen** bezeichnet.

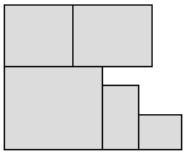

Abb. 4.2: Beispiel für einen kompaktierten Floorplan

Wie an dem Beispiel in Abbildung 4.2 zu sehen ist, sind kompaktierte Platzierungen nicht zwangsläufig frei von Lücken.

Definition 4.4

Ein Floorplan heißt **Mosaik-Floorplan**, wenn er keine Freiräume und keine Überkreuzung von zwei Schnittlinien enthält. Damit sind alle Kreuzungen von Schnittlinien im Mosaik-Floorplan T-Kreuzungen.

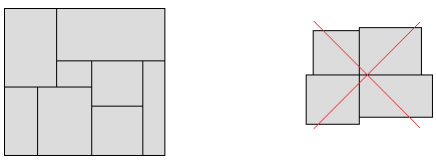

Abb. 4.3: Beispiel für einen Mosaik-Floorplan und eine Kreuzung von zwei Schnittlinien

In Abbildung 4.3 ist ein solcher Mosaik-Floorplan auf der linken Seite dargestellt. Rechts daneben ist eine Überkreuzung von Schnittlinien eingezeichnet, welche in einem Mosaik-Floorplan nicht enthalten sein darf.

Definition 4.5
Ein Floorplan, an den keine weiteren Nebenbedingungen geknüpft sind, wird **allgemeiner Floorplan** genannt.

4.2 Zweidimensionale Datenstrukturen

Eine häufig verwendete Datenstruktur für die zweidimensionale Platzierung von Rechtecken ist der **Schnittbaum**. Dieser wird durch wiederholtes Halbieren der Fläche ermittelt, deswegen sind damit nur slicing-Floorplans darstellbar.
Der Schnittbaum der Platzierung ist ein binärer Baum, dessen Blätter die Module repräsentieren und dessen Verzweigungsknoten angeben, ob der Schnitt zwischen den Modulen der beiden Teilbäume unterhalb des Verzweigungsknotens horizontal (+) oder vertikal (-) erfolgte. Der linke Teilbaum liegt in der Platzierung über bzw. links vom rechten Teilbaum.
Die in Abbildung 4.1 gezeigte slicing-Platzierung besitzt somit den folgenden Schnittbaum.

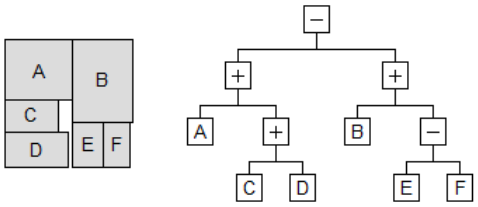

Abb. 4.4: Schnittbaum einer Platzierung

Soll mit dieser Datenstruktur aus einem gegebenen Floorplan eine neue Platzierung generiert werden, so kann zum Beispiel die Schnittrichtung geändert oder zwei Teilbäume vertauscht werden. Mit einer endlichen Anzahl dieser Operationen sind alle slicing-Floorplans von jeder beliebigen Startlösung generierbar. Der Aufwand für die Operationen liegt in $\mathcal{O}(N)$, wobei N die Anzahl der zu platzierenden Rechtecke ist.
Der große Nachteil dieser Datenstruktur ist die Beschränkung des Lösungsbereiches auf slicing-Floorplans. Damit kann nicht garantiert werden, dass die optimale Lösung mit dieser Datenstruktur gefunden wird.

Eine weitere Datenstruktur, mit der nur ein bestimmter Typ von Floorplans charakterisiert werden kann, ist die **Corner-Block-List**. Sie wurde von Xianlong Hong et al. in [HHC+00] für Mosaik-Floorplans entwickelt und besteht aus drei Sequenzen (S, L, T), welche durch sukzessives Entfernen des rechten oberen Moduls aus der Platzierung hervorgehen. In S wird die umgekehrte Reihenfolge gespeichert, in der die Module aus dem Floorplan entfernt wurden, in L deren Orientierung und in T, wie viele T-Kreuzungen unterhalb bzw. links des entfernten Moduls lagen.
Um aus einem gegeben Floorplan mit der Corner-Block-List (S, L, T) einen neuen Floorplan zu generieren, kann zum Beispiel S permutiert werden. Es ist auch möglich in L oder T eine 0 zu einer 1 (bzw. umgekehrt) zu transformieren oder in T eine zusätzliche 1 einzufügen bzw. eine zu entfernen. Allerdings muss zu einer so erzeugten Corner-Block-List kein Floorplan existieren.
Die Nachteile dieser Datenstrukturen sind zum einen die Beschränkung auf Mosaik-Floorplans und zum anderen die Unzulässigkeit generierter Codierungen. Damit kann nicht garantiert werden, dass durch sukzessive Veränderung der Startlösung die optimale Lösung erzeugt wird.

Viele Datenstrukturen arbeiten mit den **Constraint-Graphen**. Für jede Platzierung gibt es einen horizontalen und einen vertikalen Constraint-Graphen. Die Knoten repräsentieren die Module, die zu platzieren sind, außerdem wird ein Start- und ein Endknoten eingefügt. Im horizontalen Constraint-Graphen gibt es genau dann eine gerichtete Kante von einem Modul A zu einem Modul B, wenn A links von B platziert ist. Start- und Endknoten sind zu allen Knoten adjazent. Analog werden die gerichteten Kanten im vertikalen Constraint-Graphen gebildet.
Der Constraint-Graph kann ohne Informationsverlust vereinfacht werden, indem alle transitiven Kanten entfernt werden. Transitive Kanten sind gerichtete Kanten von einem Knoten A zu einem Knoten B, zwischen denen auch ein gerichteter Weg von A nach B existiert.

Aus den Constraint-Graphen lassen sich alle Informationen direkt ablesen. Entspricht das Kantengewicht der Breite bzw. Höhe des Anfangsknotens der Kante, so kann damit über den längsten gerichteten Weg zwischen Start- und Endknoten die Breite bzw. Höhe der Fläche des Floorplans ermittelt werden. Der längste gerichtete Weg

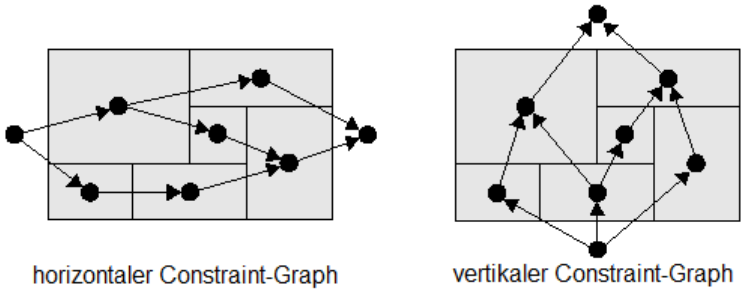

Abb. 4.5: Platzierung mit eingezeichnetem vereinfachten horizontalen und vertikalen Constraint-Graphen

zwischen dem Startknoten und einem anderen Knoten gibt die x- bzw. y-Koordinate der linken unteren Ecke des entsprechenden Moduls an. Damit lassen sich alle kompaktierten und viele allgemeine Platzierungen eindeutig charakterisieren.
Die Constraint-Graphen als Datenstruktur haben allerdings grundlegende Nachteile: Es gibt sehr viele Floorplans, die damit charakterisiert werden können, praktisch aber keinerlei Nutzen haben. Das sind alle die Floorplans, in denen zwei Module sowohl über als auch nebeneinander zu platzieren sind. In diesem Fall schließt das zweite Modul an der rechten oberen Ecke des ersten an. Ein Beispiel für eine solche ungünstige Platzierung wird in Abbildung 4.6 dargestellt.

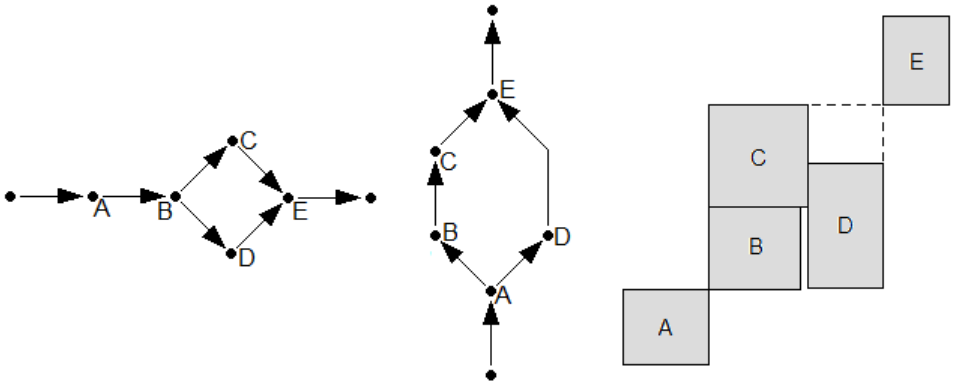

Abb. 4.6: Constraint-Graphen mit resultierender ungünstiger Platzierung

Ein weiterer Nachteil ist, dass nicht zu allen Constraint-Graphen eine zulässige Platzierung existiert. Gibt es zwei Module X und Y, zwischen denen weder im horizontalen noch im vertikalen Constraint-Graphen eine Kante vorliegt, so gibt es zwischen diesen Modulen keine Relation und es können im Floorplan Überschneidungen entstehen. Das ist aber nicht zulässig.
In Abbildung 4.7 ist ein Paar von Constraint-Graphen und die daraus resultierende

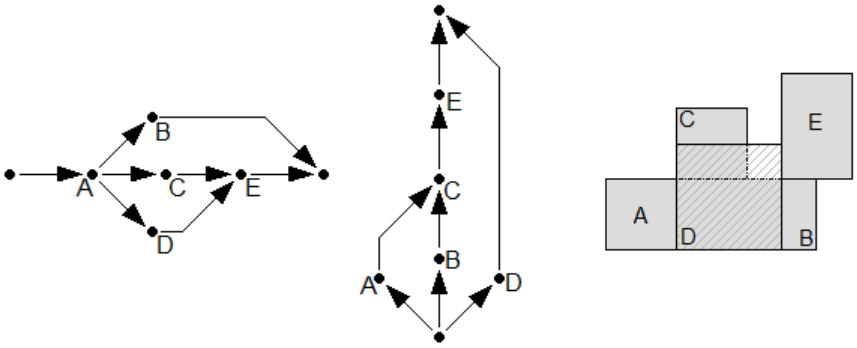

Abb. 4.7: Constraint-Graphen mit resultierender unzulässiger Platzierung

unzulässige Platzierung dargestellt. Es ist zu sehen, dass in den Constraint-Graphen zwischen den Modulen B und D sowie C und D keine Relation codiert ist. Deswegen kommt es auch zu einer Überschneidung. Durch eine Abfrage könnte zwar zwischen allen Modulen eine Relation geschaffen werden, indem zum Beispiel im vertikalen Constraint-Graphen aus Abbildung 4.7 eine Kante zwischen dem Modul C und dem Modul D eingefügt wird, dazu müsste aber für jedes Paar von Modulen geprüft werden, ob eine Relation vorliegt.
Außerdem muss darauf geachtet werden, dass keine gerichteten Kreise in den Constraint-Graphen vorliegen, da dann ein Modul gleichzeitig über und unter (bzw. links und rechts) von einem anderen Modul zu platzieren wäre.
Trotzdem sind die Constraint-Graphen wichtig für viele andere Datenstrukturen, da sie den Übergang zwischen der Datenstruktur und dem Floorplan ermöglichen. Zu diesem Zweck werden die Constraint-Graphen auch in der Datenstruktur Sequenz-Paar verwendet.

Eine andere Datenstruktur ist der von Pei-Ning Guo, Chung-Kuan Cheng und Takeshi Yoshimura in [GCY99] vorgestellte **O-Tree**. Damit lassen sich viele allgemeine Floorplans darstellen. Der O-Tree ist ein geordneter Baum, dessen Knoten die Module repräsentieren. Die Wurzel des Baumes ist ein zusätzlicher Knoten, der den linken Rand symbolisiert. Zwischen zwei Modulen X und Y muss eine Kante eingefügt werden, wenn X das unterste Modul ist, an das Y links anschließt. Die Ordnung des Baumes beschreibt die vertikale Ordnung der Module.

In der Abbildung 4.8 ist ein Floorplan mit dem O-Tree eingezeichnet. Um Speicher

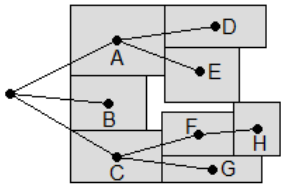

Abb. 4.8: Floorplan mit eingezeichnetem O-Tree

zu sparen, wird der O-Tree als Tupel (T, Π) gespeichert. Dabei ist T eine binäre Liste, die angibt, wie der Baum bei einer Tiefensuche durchlaufen wurde. Wurde eine Kante in Richtung Blatt durchlaufen, so wird eine 0 eingetragen, anderenfalls eine 1. In Π werden die Module in der Reihenfolge eingetragen, in der sie bei der Tiefensuche erreicht wurden.
Durch eine Permutation von T oder Π kann ein neuer O-Tree erzeugt werden. Aus dem generierten Baum können dann in $\mathcal{O}(N)$ die beiden Constraint-Graphen und damit der Floorplan ermittelt werden.
Ein Nachteil dieser Methode ist, dass nicht alle erzeugten Bäume eine Lösung codieren.

Es gibt noch weitere Datenstrukturen für die Platzierung von Rechtecken in der Ebene. Diese sollen in dieser Arbeit aber nicht näher erläutert werden.

4.3 Sequenz-Paar

4.3.1 Grundlagen

Die Datenstruktur Sequenz-Paar wurde in [MFNK96] von H. Murata et al. beschrieben. Es werden zwei Sequenzen der N zu platzierenden Module gespeichert, aus denen die Beziehung zweier Module direkt abgelesen werden kann. Liegen zwei Module in beiden Sequenzen in derselben Reihenfolge vor, so sind sie nebeneinander platziert – unterscheidet sich die Reihenfolge in beiden Sequenzen, so liegen die Module in der Platzierung übereinander.

$$(\ldots A \ldots B \ldots), (\ldots A \ldots B \ldots) \implies A \text{ ist links von } B \text{ platziert}$$
$$(\ldots B \ldots A \ldots), (\ldots A \ldots B \ldots) \implies A \text{ ist unterhalb von } B \text{ platziert}$$

Zur Verbesserung der Lesbarkeit werden die Beziehungen zweier Module im Folgenden durch diese Symbolik wiedergegeben:

- $A \uparrow B$: A liegt unterhalb von B
- $A \rightarrow B$: A liegt links von B
- $A \downarrow B$: A liegt oberhalb von B
- $A \leftarrow B$: A liegt rechts von B

Ableitung des Sequenz-Paars aus dem Floorplan

Um aus einem gegebenen Floorplan das Sequenz-Paar zu generieren, wird zunächst in die Mitte eines jeden Moduls ein Startpunkt gesetzt und anschließend von diesem aus in der im Folgenden erläuterten Art und Weise je eine Linie zu jeder Ecke des Floorplans eingezeichnet. Die Reihenfolge, in der die Linien an den Ecken ankommen, bestimmt die Reihenfolge der Module im Sequenz-Paar.

Die erste Sequenz wird aus der Reihenfolge der Linien an der linken unteren bzw. der rechten oberen Ecke ermittelt. An beiden Ecken müssen die Linien in derselben Reihenfolge vorliegen, sonst ist der gegebene Floorplan nicht codierbar.

Die Linien zur rechten oberen Ecke sind wie folgt zu ziehen: Zuerst wird ausgehend vom Startpunkt in der Mitte des Moduls eine Linie nach rechts gezogen (das grüne Segment 1 in Abbildung 4.9), bis das nächste Modul erreicht ist. Von dort an wird die Linie bis zum nächsten Modul nach oben fortgesetzt (Segment 2, rot). Dann ist die Linie wieder vertikal nach rechts bis zum nächsten Modul weiter zu zeichnen. Auf diese Art wird solange abwechselnd nach rechts und nach oben die Linie fortgesetzt, bis der Rand des Floorplans erreicht ist und ein letztes Liniensegment zur rechten oberen Ecke zu zeichnen ist. Die Linien zweier Module müssen dabei so gezeichnet werden, dass sie sich nicht überkreuzen.

Auf analogem Weg sind die Linien zur linken unteren Ecke einzuzeichnen, abwechselnd nach links und nach unten. An beiden Ecken sollten die Linien in derselben Reihenfolge vorliegen. Diese Reihenfolge bildet die erste Sequenz und ergibt sich hier zu $(ADBCEFG)$.

Für die zweite Sequenz werden die Linien mit einem vertikalen Segment begonnen und wieder treppenartig die linke obere und rechte untere Ecke angestrebt. Damit

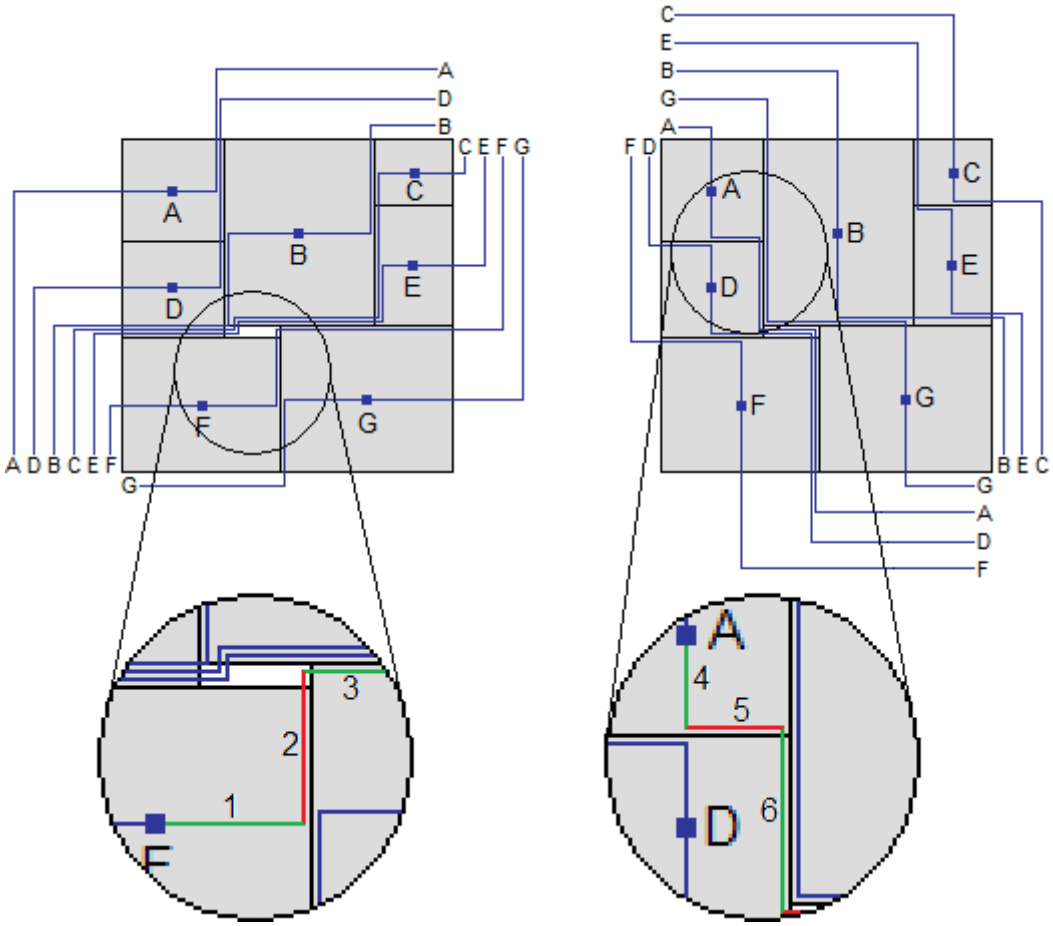

Abb. 4.9: Ermittlung des Sequenz-Paares eines Floorplans

ergibt sich $(FDAGBEC)$ für die zweite Sequenz in dem Beispiel aus Abbildung 4.9. Das Sequenz-Paar des Beispiels lautet also $(ADBCEFG), (FDAGBEC)$.
Diese Vorgehensweise ist sehr aufwändig, muss aber in der Praxis nicht angewendet werden, da diese Richtung der Codierung nicht vollzogen wird. Vielmehr wird durch Permutation ein neues Sequenz-Paar erzeugt und dieses dann in eine Platzierung umgewandelt.

Mit dieser Methode kann jeder kompaktierte Floorplan und viele allgemeine Platzierungen in ein Sequenz-Paar umgewandelt werden.
Es gibt aber auch Platzierungen, die nicht mit dem Sequenz-Paar darstellbar sind. Floorplans mit Lücken, die sich nicht aus einer der oben genannten Beziehungen ergeben, sind zum Beispiel nicht mit dem Sequenz-Paar eindeutig codierbar. Es ist zwar meist möglich, das entsprechende Sequenz-Paar zu bilden, allerdings verschwinden die Lücken und damit ändert sich die Platzierung bei der Rücktransformation.

Auch Platzierungen, deren Bedingungen nicht transitiv sind, können nicht dargestellt werden. Die Bedingungen sind transitiv, wenn aus $X \sim Y$ und $Y \sim Z$ stets folgt, dass $X \sim Z$ ist, wobei $\sim$ in der Kette genau eine Beziehung repräsentiert. Das muss aber nicht unbedingt der Fall sein, wenn Freiräume in der Platzierung enthalten sein sollen.

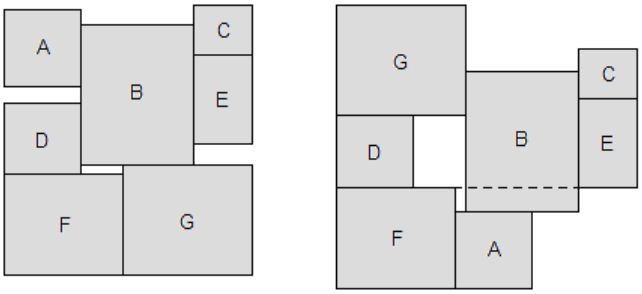

Abb. 4.10: Gegenbeispiele für die Codierbarkeit von Platzierungen mit dem Sequenz-Paar

Bei den beiden Floorplans in Abbildung 4.10 wurden unnötige Freiflächen eingefügt. Bei dem linken Gegenbeispiel würde das Einzeichnen der Linien zu dem Sequenz-Paar des Floorplans aus Abbildung 4.9 führen. Nach der Rücktransformation, also der Umwandlung des Sequenz-Paars in eine Platzierung, wären die Freiräume aber nicht mehr vorhanden, da sie nicht mit einer der Platzierungsregeln, wie zum Beispiel „Modul X liegt über Modul Y“, erklärt werden können. Da die Lücken verloren gehen, kann behaupten werden, dass dieser Floorplan so nicht codierbar ist.
Bei dem rechten Gegenbeispiel hingegen kann der Freiraum durch eine Beziehung zweier Module zueinander angegeben werden. Die Lücke entsteht, wenn das Modul E über dem Modul F zu platzieren ist. Diese Bedingung führt hier aber zu einem Widerspruch:
Wie in dem Floorplan zu sehen, gilt $F \rightarrow B$ und $B \rightarrow E$, daraus folgt für diese Module im Sequenz-Paar $(\ldots F \ldots B \ldots E \ldots), (\ldots F \ldots B \ldots E \ldots)$.
Die zweite Bedingung, die den Freiraum unter E zur Folge hat, lautet wie bereits erwähnt $F \uparrow E$, was durch folgende Anordnung im Sequenz-Paar zu codieren ist: $(\ldots E \ldots F \ldots), (\ldots F \ldots E \ldots)$.
Es ist also offensichtlich nicht möglich, diese beiden Forderungen mit einem Sequenz-Paar zu erfüllen.
Dieser Floorplan kann also nicht mit der Datenstruktur Sequenz-Paar codiert werden.

Ermittlung des Floorplans aus dem Sequenz-Paar

Soll der Floorplan von Hand aus einem gegebenen Sequenz-Paar erzeugt werden, so kann wie folgt vorgegangen werden:

Zunächst wird ein Raster gezeichnet, welches in jede Richtung N sich kreuzende Linien aufweist und auf eine Ecke gedreht wurde. An die links und rechts unten endenden Linien werden die Module in der Reihenfolge des Sequenz-Paars notiert. Damit wird jedem Modul ein sich kreuzendes Linienpaar zugeordnet. Auf dem Kreuzungspunkt dieser Linien werden die Module vorplatziert. Anschließend wird alles soweit wie möglich nach links unten verschoben und Überlappungen entfernt. Dabei ist darauf zu achten, dass die Platzierung nicht verfälscht wird. Es müssen alle Module, die sich in dem Quadranten oberhalb des Rechtecks X befinden, in dem Floorplan über X platziert sein. Analog verhält es sich mit den anderen Quadranten.

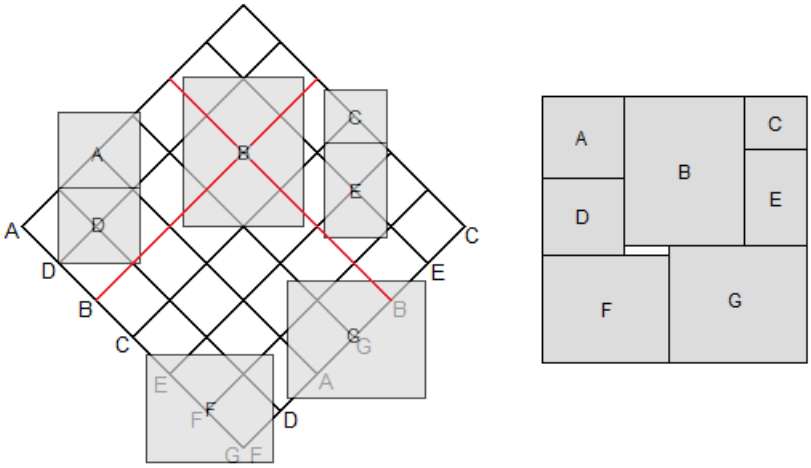

Abb. 4.11: Ermittlung des dem Sequenz-Paar $(ADBCEFG), (FDAGBEC)$ zugehörigen Floorplans

Für Modul B sind in Abbildung 4.11 die vier Quadranten eingezeichnet. Aus diesen kann abgeleitet werden, dass oberhalb von B kein Modul zu platzieren ist. Rechts davon müssen die Module C und E liegen, unterhalb von B die Module F und G und schließlich sind links neben B die Module A und D zu platzieren.

Da die beiden Module A und D im oberen Quadranten des Moduls F liegen, müssen diese auch oberhalb von F platziert werden, nicht, wie aufgrund der Zeichnung fälschlicherweise angenommen werden könnte, links von F.

Der Floorplan kann auch anders aus dem Sequenz-Paar abgeleitet werden, indem zunächst die Constraint-Graphen erzeugt und im Anschluss die linke untere Ecke jedes Moduls explizit berechnet wird.
Um den horizontalen Constraint-Graphen zu erstellen, wird zunächst für jeden Modul X die Menge der linken und rechten Module $M_l(X)$ und $M_r(X)$ ermittelt. Wie schon erwähnt, gibt die Reihenfolge zweier Module in den beiden Sequenzen deren Beziehung an. Ein Modul A ist links von B platziert, wenn A in beiden Sequenzen vor B steht. Damit gilt: $A \in M_l(B)$ und $B \in M_r(A)$. Die beiden Mengen ergeben sich also zu:

$$M_l(X) = \{Y \mid (\ldots Y \ldots X \ldots), (\ldots Y \ldots X \ldots)\}$$
$$M_r(X) = \{Y \mid (\ldots X \ldots Y \ldots), (\ldots X \ldots Y \ldots)\}$$

Mit diesen beiden Mengen für jeden Knoten lässt sich der horizontale Constraint-Graph erzeugen:
Als erstes werden die Knoten erstellt. Es wird ein Startknoten s, ein Knoten x für jeden Modul X und ein Endknoten t benötigt.
Danach sind die gerichteten Kanten einzufügen:

- für jeden Knoten v gibt es eine Kante von s zu v und eine Kante von v zu t:
  $E = \{(s, v) \mid v \in V \setminus \{s, t\}\} \cup \{(v, t) \mid v \in V \setminus \{s, t\}\}$.
- für jedes Knotenpaar u, v mit den zugehörigen Modulen U, V gibt es genau dann eine Kante (u, v), wenn $U \in M_l(V)$ und damit auch $V \in M_r(U)$:
  $E = E \cup \{(u, v) \mid U \in M_l(V)\}$.

Abschließend werden den Kanten noch Gewichte gegeben: Sei $e = (u, v)$ eine gerichtete Kante und $a(U)$ die Breite des Moduls U, das dem Knoten u zugeordnet ist. Dann ist das Kantengewicht wie folgt zu setzen: $w((u, v)) = a(U)$. Start- und Endknoten haben keine zugeordnetes Modul und damit eine Breite von Null.
Der Constraint-Graph enthält keine gerichteten Kreise, da sonst ein Modul gleichzeitig über und unter (bzw. rechts und links) von einem anderen Modul zu platzieren wäre.
Analog wird beim horizontalen Constraint-Graphen vorgegangen. Allerdings sind die hier entscheidenden Mengen die der Module, die ober- bzw. unterhalb eines Moduls X zu platzieren sind:

$$M_o(X) = \{Y \mid (\ldots Y \ldots X \ldots), (\ldots X \ldots Y \ldots)\}$$
$$M_u(X) = \{Y \mid (\ldots X \ldots Y \ldots), (\ldots Y \ldots X \ldots)\}$$

Die Knoten sind wie im horizontalen Constraint-Graphen zu erstellen, die Kanten mit dem Start- und Endknoten auf dieselbe Weise einzufügen. Außerdem sind für jeden Knoten v eines Moduls V gerichtete Kanten ausgehend von den Knoten der Module aus $M_u(V)$ einzufügen. Die Kanten zu den Knoten der Module aus $M_o(V)$ ergeben sich wieder automatisch. Die Kantengewichte werden nun mit den Höhen der Module ermittelt: Sei $b(U)$ die Höhe des Moduls U, das dem Knoten u zugeordnet ist. Dann ist $w((u, v)) = b(U)$. Die Höhe von Start- und Endknoten ist wieder mit Null gleichzusetzen.

Um nun aus den beiden Constraint-Graphen die Koordinaten der linken unteren Ecke eines jeden Moduls und die Breite und Höhe der Fläche des Floorplans zu ermitteln, ist ein Algorithmus nötig, der den längsten gerichteten Weg in einem gerichteten kreisfreien Graphen mit Kantengewicht findet.
Ein sehr einfacher Algorithmus zur Bestimmung der längsten Wege eines azyklischen Graphen ist der folgende **rekursive Markierungsalgorithmus**:
Im ersten Schritt müssen die Quelle und die Senke bestimmt werden. Die Quelle ist ein Knoten, in dem keine gerichtete Kante endet, die Senke ist ein Knoten, von dem keine gerichtete Kante ausgeht. Damit ist der Startknoten s die Quelle und der Endknoten t die Senke.
Jetzt sind die Knoten zu markieren. Es wird mit der Senke begonnen, deren Markierung sich wie folgt berechnet: Sei $N_V[t] = \{u \mid \exists(u, t) \in E\}$ die Menge der Vorgänger des Knotens t, dann ergibt sich dessen Markierung zu:

$$m(t) = \max\{m(u) + w((u, t)) \mid u \in N_V[t]\}.$$

Der Knoten t bekommt damit als Markierung die Länge des längsten Weges von der Quelle zur Senke. Sind nun aber die Vorgängerknoten von t noch nicht markiert, so wird deren Markierung auf dieselbe Art errechnet. Sei v ein unmarkierter Vorgängerknoten von t, dann berechnet sich dessen Markierung mit der Formel $m(v) = \max\{m(u) + w((u, v)) \mid u \in N_V[v]\}$. Sind wiederum Vorgänger von v unmarkiert, so wird deren Markierung auf die gleiche Weise ermittelt. So ergibt sich eine rekursive Berechnung der Markierung aller Knoten, bei der jeder Knoten und jede Kante genau einmal untersucht werden müssen. Damit beträgt der Aufwand $\mathcal{O}(|V| + |E|)$.
Die x- und y-Koordinate der linken unteren Ecke eines Moduls ergibt sich aus der

Markierung des zugehörigen Knotens im horizontalen und im vertikalen Constraint-Graphen. Die Breite und Höhe des Floorplans entspricht der Markierung von t.

4.3.2 Anwendung im dreidimensionalen Fall

Mit dem Sequenz-Paar kann die Platzierung von N Modulen, gespeichert in der Menge $\mathcal{B}$, in der Ebene dargestellt werden. Es wird aber eine Datenstruktur benötigt, welche die Platzierung der Module auf k Ebenen darstellen kann. Diese quasi-dreidimensionale Platzierung hat einige Besonderheiten, welche die Codierung erleichtern:

- jedes Modul ist auf genau einer Ebene zu platzieren
- ein Modul ragt nicht über seine Ebene hinaus
- die Platzierung auf einer Ebene beeinflusst nicht die Platzierung auf anderen Ebenen.

Damit kann die Platzierung auf k Ebenen mit einem k-Tupel von Sequenz-Paaren $(S_1, S_1, \ldots, S_k)$ dargestellt werden. Jedes Sequenz-Paar S_i codiert die Platzierung der Module der Ebene $\mathcal{B}_i \subset \mathcal{B}$. Es muss gelten:

$$\bigcup_{i=1}^{k} \mathcal{B}_i = \mathcal{B}$$
$$\forall i = 1 \ldots k : \mathcal{B}_i \neq \emptyset$$
$$\forall i, j = 1, \ldots k, i \neq j : \mathcal{B}_i \cap \mathcal{B}_j = \emptyset.$$

Damit die Ebenen zentrisch übereinander liegen, wird der Koordinatenursprung jeder Ebene in die Mitte gesetzt und die Positionen der Module werden angepasst:
Sei a_i die Breite und b_i die Höhe der Ebene i. Für jeden Modul wurde die Position (x, y) der linken unteren Ecke ermittelt. Diese ergibt sich nun zu $(x - \frac{a_i}{2}, y - \frac{b_i}{2})$.

4.3.3 Die Operationen

Ziel der Codierung der Platzierung ist es, durch einfache Operationen neue Platzierungen zu erhalten. Mit diesen Operationen sollten alle Lösungen erreichbar sein.

1. Vertauschen zweier Module

Die erste Operation ist das Vertauschen zweier Module. Dabei sollen die beiden Module unmittelbar ihre Position in der Anordnung tauschen.

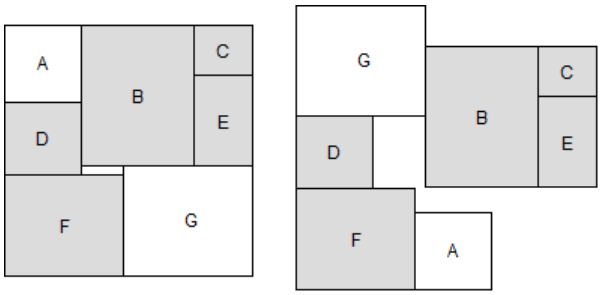

Abb. 4.12: Beispiel: Vertauschen zweier Module

Zwei Module tauschen ihre Position, wenn sie in beiden Sequenzen vertauscht werden. Das Sequenz-Paar des Beispiels lautet $(ADBCEFG), (FDAGBEC)$. Es werden die Module A und G vertauscht: $(GDBCEFA), (FDGABEC)$. Mit diesem neuen Sequenz-Paar wird die rechte Platzierung aus Abbildung 4.12 codiert. Es ist zu beachten, dass der Modul B über dem Modul F platziert werden muss und deswegen nicht bündig auf dem Modul A aufsitzt.

Auf dieselbe Weise können auch Module verschiedener Ebenen vertauscht werden. Die zwei Module werden jeweils auf ihrer Ebene gelöscht und im Sequenz-Paar der neuen Ebene an den Positionen des anderen Moduls eingefügt: $(ABC), (BCA)$ und $(DEF), (FDE)$ seien die Sequenz-Paare zweier Ebenen. Die Module A und D sollen vertauscht werden, wodurch die Sequenz-Paare $(DBC), (BCD)$ und $(AEF), (FAE)$ entstehen.

2. Veränderung der Beziehung zweier Module

Ein Modul kann oberhalb oder unterhalb, rechts oder links von einem anderen Modul platziert sein. Sie stehen dann in entsprechender Beziehung. Um diese Relation von übereinander auf nebeneinander (und umgekehrt) zu ändern, ist das Vertauschen der Module in nur einer Sequenz notwendig.

Wird im obigen Beispiel F und B in der ersten Sequenz vertauscht, entsteht das Sequenz-Paar $(GDFCEBA), (FDGABEC)$. Die zugehörige Platzierung hat dann die in Abbildung 4.13 gezeigte Gestalt.

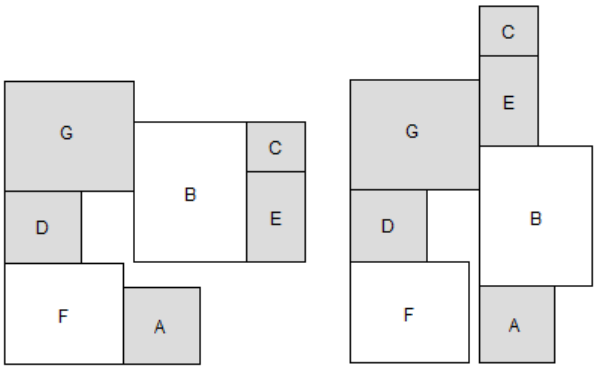

Abb. 4.13: Beispiel: Verändern der Beziehung zweier Module

Wie in 4.13 zu sehen ist, ändern sich damit auch die Beziehungen zu den anderen Modulen. Durch diese Operation wird der Floorplan deutlich stärker verändert.

3. Verschieben eines Moduls auf eine andere Ebene

Soll ein Modul auf eine andere Ebene verschoben werden, so muss es auf der eigenen Ebene entfernt und in der neuen eingefügt werden. Für das Einfügen gibt es zwei Möglichkeiten:

- Das Modul kann in beiden Sequenzen hinten angefügt werden. Damit wird es am rechten Rand platziert.
- Der Modul kann in beiden Sequenzen an einer zufälligen Stelle eingefügt werden und wird damit zwischen den anderen Modulen positioniert.

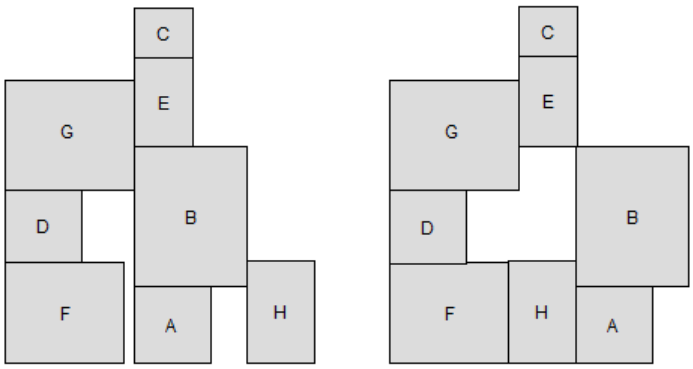

Abb. 4.14: Beispiel: Einfügen eines Moduls

Das linke Beispiel der Abbildung 4.14 zeigt das Einfügen am Ende der beiden Sequenzen. Das zugehörige Sequenz-Paar ergibt sich zu $(GDBCEFAH), (FDGABECH)$.

Auf der rechten Seite der Abbildung wird das zufällige Einfügen demonstriert, mit $(GDBCEHFA), (FDHGABEC)$ als Sequenz-Paar.

4. Verändern des Seitenverhältnisses eines Moduls

Da die Größe der Superzellen, die hier platziert werden sollen, nur eine Schätzung ist und diese aus vielen kleinen Zellen bestehen, sind die Seitenlängen variabel. Es ist also möglich, eine Superzelle zu verzerren, indem das Seitenverhältnis bei gleichbleibender Fläche geändert wird: $g(v) = (a, b) \Rightarrow g(v) = (a_{neu}, b_{neu})$, wobei gilt: $a \cdot b = a_{neu} \cdot b_{neu}$ und $\frac{1}{2} \leq \frac{a_{neu}}{b_{neu}} \leq 2$. Damit bleibt die Fläche, die das Modul einnimmt, unverändert. Mit der letzten Bedingung wird das Entstehen sehr schmaler Module verhindert, indem verboten wird, dass eine Seite mehr als doppelt so lang wie die andere ist. Um einen endlichen Lösungsraum zu erhalten, wurde eine endliche Menge zulässiger Seitenverhältnisse $S = \{s_1, s_2, \ldots, s_p\}$, $\forall i = 1, 2, \ldots, p : \frac{1}{2} < s_i < 2$ bestimmt. Für jeden Modul muss gelten: $\exists s_i \in S : \frac{a}{b} = s_i$.

4.3.4 Polynomial zulässiger Lösungsraum und Vollständigkeit der Operationen

Eine Datenstruktur sollte den Lösungsraum möglichst vollständig wiedergeben können. Sie ist nur dann in der Praxis geeignet, wenn aus der Codierung in polynomialer Zeit eine Lösung abgeleitet werden kann. Auch sollte die optimale Lösung codiert werden können.
Soll die optimale Lösung durch sukzessives Verändern einer Startlösung ermittelt werden, so muss durch die Wahl der Operationen, die eine neue Lösung generieren, gewährleistet sein, dass ausgehend von jeder Lösung jede andere erreichbar ist.
Im Folgenden soll die Eignung der Datenstruktur Sequenz-Paar anhand dieser Kriterien gezeigt werden.

Definition 4.6

Der von einer Datenstruktur erzeugte Lösungsraum heißt **p-zulässig**, wenn folgende Bedingungen erfüllt sind:

- der Lösungsraum ist endlich
- jede Lösung ist realisierbar

- die Realisierung einer Lösung ist in polynomialer Zeit durchführbar
- es gibt in der Datenstruktur eine Darstellung für die optimale Lösung.

Satz 4.1. *Die hier vorgestellete Datenstruktur Sequenz-Paar, angewandt auf das quasi-dreidimensionale Problem der Platzierung von N Superzellen auf k Ebenen, spannt einen p-zulässigen Lösungsraum auf.*

Beweis:

Der Lösungsraum ist endlich

Es gibt nur endlich viele k-Tupel von Sequenz-Paaren, die alle N Superzellen genau einmal enthalten:

Zunächst einmal müssen die N Superzellen aus $\mathcal{B}$ auf die k Ebenen aufgeteilt werden. Das entspricht einer Partition der Menge in k Teile. Die Menge all dieser Partitionen sei $P_k(\mathcal{B}) = \{P = \{\mathcal{B}_1, \ldots, \mathcal{B}_k\}\}$. Es gilt $|P_k(\mathcal{B})| = S_{N,k}$, wobei $S_{N,k}$ die Stirling-Zahl 2. Art ist. Jede dieser Partitionen kann auf $k!$ Möglichkeiten den Ebenen zugeordnet werden. Somit existieren $S_{N,k} \cdot k!$ Aufteilungen von N Superzellen auf k Ebenen.

Für jede Ebene i, auf der sich $|\mathcal{B}_i|$ Module befinden, gibt es $(|\mathcal{B}_i|!)^2$ Sequenz-Paare: $|\mathcal{B}_i|!$ Permutationen für jede der beiden Sequenzen.

Insgesamt gibt es also $\sum_{P_k(\mathcal{B})} k! \cdot \prod_{i=1}^{k} (|\mathcal{B}_i|!)^2$ Sequenz-Paar-k-Tupel.

Darin nimmt jedes Modul eines der p Seitenverhältnisse aus S an, sodass $p^{|\mathcal{B}|}$ Möglichkeiten für die Seitenverhältnisse existieren. Es gibt also

$$p^{|\mathcal{B}|} \cdot \sum_{P_k(\mathcal{B})} k! \cdot \prod_{i=1}^{k} (|\mathcal{B}_i|!)^2$$

und damit endlich viele Lösungen für das Platzierungsproblem mit der Datenstruktur Sequenz-Paar unter den getroffenen Annahmen.

Jede Lösung ist realisierbar

Es ist zu zeigen, dass es zu jedem Sequenz-Paar eine zulässige Lösung gibt, also eine eindeutige Anordnung der Module ohne Überschneidungen.

Wie in Kapitel 4.3.1 gezeigt, lässt sich aus einem Sequenz-Paar über die Constraint-Graphen ein Floorplan erzeugen. In diesem kann es nicht zu Überschneidungen kommen, da für jedes Paar von Modulen eine eindeutige Beziehung, übereinander

oder nebeneinander, aus dem Sequenz-Paar resultiert und ein Modul die Position zugeordnet bekommt, die gewährleistet, dass es keine Überschneidung zu einem Modul unterhalb und links von diesem gibt.

Der Aufwand der Realisierung einer Lösung ist polynomial

Da das Erstellen der Constraint-Graphen aus $\mathcal{O}(N^2)$ und das Bestimmen der linken unteren Ecke aller Module aus $\mathcal{O}(N + N^2)$ ist, ist die Lösung in $\mathcal{O}(N^2)$ und damit in polynomialer Zeit aus dem Sequenz-Paar ableitbar.

Die optimale Lösung ist darstellbar

Mit dem Sequenz-Paar sind viele, aber nicht alle Platzierungen darstellbar. Es lassen sich aber alle kompaktierten Floorplans mit dem Sequenz-Paar codieren:
Ist eine Platzierung kompaktiert, so kann kein Modul nach links oder nach unten verschoben werden, ohne dass ein anderes Modul ebenfalls verschoben werden müsste, um Überschneidungen zu verhindern. Damit kann die Position eines jeden Moduls im Floorplan eindeutig durch die Beziehungen zu seinen unmittelbaren Nachbarn angegeben werden. Deswegen gibt es keine nicht-transitiven Beziehungen, die zur Beschreibung des Floorplans notwendig wären, da sonst ein nicht unmittelbar benachbarter Modul eine Forderung stellen würde: Sei $A \uparrow C$ und $A \rightarrow B$, $B \rightarrow C$. Dann kann A kein unmittelbarer Nachbar sein, weil B zwischen A und C liegt. Da der Floorplan kompaktiert ist, liegt C soweit unten wie möglich und die Forderung $A \uparrow C$ kann zur Beschreibung des Floorplans entfallen.
Damit ist die Charakterisierung des Floorplans ausschließlich mit transitiven Forderungen möglich und aus diesen ein Sequenz-Paar ableitbar.
Es bleibt zu zeigen, dass die optimale Lösung auch kompaktiert ist. Dazu muss zunächst ein Optimalitätskriterium festgelegt werden. Das Platzierungsproblem im Chip-Entwurf sucht primär eine Lösung mit kleiner Fläche – große Lücken und Freiräume sind keinem Optimierungsziel dienlich. Deswegen wird der optimale Floorplan immer auch kompaktiert sein.

Damit ist gezeigt, dass die Datenstruktur Sequenz-Paar mit einer endlichen Zahl möglicher Seitenverhältnisse einen p-zulässigen Lösungsraum aufspannt.

□

Satz 4.2. *Die Menge der Operationen aus Kapitel 4.3.3 ist vollständig. Das heißt, von jeder beliebigen Startlösung aus kann jede beliebige Ziellösung mit einer endlichen Anzahl dieser Operationen erreicht werden.*

Beweis:
Da es zu jeder Operation eine Umkehrung gibt, reicht der Beweis, dass aus jeder beliebigen Lösung eine Basislösung erreicht werden kann. Diese Basislösung sei eine Kette der quadratischen Module in einer gegebenen Reihenfolge auf Ebene 1. Alle anderen Ebenen seien leer. Das dazugehörige Sequenz-Paar-k-Tupel lautet: $(A, B, C, \ldots, N), (A, B, C, \ldots, N)$ $(), () \ldots (), ()$.
Ausgehend von einem beliebigen Floorplan wird die Basislösung folgendermaßen erreicht:

1. Verschiebe alle Module auf die unterste Ebene
2. $\forall i = 1 \ldots N$: Vertausche in der ersten Sequenz das i-te Modul mit dem Modul, das in der Basislösung das i-te Modul ist
3. $\forall i = 1 \ldots N$: Vertausche in der zweiten Sequenz das i-te Modul mit dem Modul, das in der Basislösung das i-te Modul ist
4. Setze das Seitenverhältnis aller Module auf 1.

□

Damit ist die Datenstruktur Sequenz-Paar-k-Tupel mit den erläuterten Operationen zum Finden der optimalen Platzierung geeignet.

4.4 Dreidimensionale Datenstrukturen

Eine dreidimensionale Datenstruktur macht vor allem dann Sinn, wenn es dreidimensionale Zellen gibt. Bei den in dieser Arbeit betrachteten Schaltkreisen sind aber nur die TSVs wirklich dreidimensional, da sie über verschiedene Ebenen gehen. Alle anderen Zellen werden auf genau einer Ebene platziert, sodass sie mit einer quasi-dreidimensionalen Datenstruktur – wie sie in den vorangegangenen Abschnitten vorgestellt wurde – gut platziert werden können. Es gibt aber auch Schaltkreise mit Bauteilen, die höher als eine Ebene sind und damit auf mehreren Ebenen platziert werden müssen. Deswegen werden vermehrt Datenstrukturen entwickelt, in denen

die dritte Dimension explizit berücksichtigt wird. Eine solche Datenstruktur soll im Folgenden kurz vorgestellt werden.

Eine der ersten 3D-Datenstrukturen ist das Sequenz-Quintupel. Diese Datenstruktur wurde von Hiroyuki Yamazaki et al. [YSNK00] entwickelt und ist eine Erweiterung des Sequenz-Paars auf die dritte Dimension.
Im Sequenz-Quintupel werden mit den ersten beiden Sequenzen die Beziehungen $X \rightarrow Y$ und $X \leftarrow Y$ codiert. Aus diesen Sequenzen kann dann der vertikale Constraint-Graph abgeleitet werden, indem genau dann eine Kante zwischen zwei Modulen eingefügt wird, wenn sie in beiden Sequenzen in derselben Reihenfolge vorliegen. Analog wird aus der dritten und vierten Sequenz der horizontale Constraint-Graph ermittelt. Dann werden die x-y-Koordinaten aller Module aus den Constraint-Graphen mit Hilfe der längsten Wege berechnet. Dabei kann es zu Überschneidungen kommen, da nicht alle Module in einer Relation zueinander stehen. Aus der letzten Sequenz wird die z-Koordinate abgeleitet, indem ein dritter Constraint-Graph erstellt wird. In diesem werden die Module durch eine Kante verbunden, deren x-y-Koordinaten sich überschneiden. Die Richtung der Kante wird durch die letzte Sequenz bestimmt.

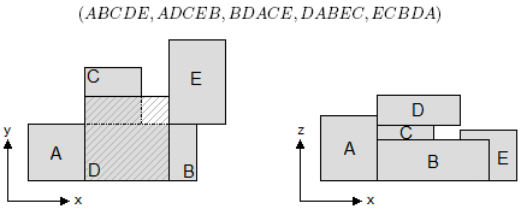

Abb. 4.15: 3D-Floorplan und zugehöriges Sequenz-Quintupel

In Abbildung 4.15 ist der aus $(ABCDE, ADCEB, BDACE, DABEC, ECBDA)$ resultierende Floorplan aus zwei Perspektiven dargestellt. In der x-y-Perspektive wurde die Überschneidung der Module B und C mit dem Modul D durch die Schraffur dargestellt. Die x-z-Ansicht zeigt den Floorplan, nachdem alle Sequenzen ausgewertet und die Überschneidungen durch Anpassung der z-Koordinate beseitigt wurden. Es ist zu sehen, dass Modul D oberhalb von B und C zu platzieren ist.
Mit dem Sequenz-Quintupel können sehr viele 3D-Platzierungen dargestellt werden. Allerdings gibt es mit dieser Codierung sehr viele unnötige Informationen. So kann die letzte Sequenz in dem Beispiel auch $CABED$ oder $ABCDE$ lauten, die einzig wichtige Information, die damit codiert wird, ist die Relation zwischen

den Module B, C und D. Aber auch nach verschiedenen Änderungen in den anderen vier Sequenzen bleibt der Floorplan erhalten. Werden zum Beispiel die Module A und D in Sequenz 3 und 4 vertauscht, findet keine Veränderung am Floorplan statt.

Es gibt neben dem Sequenz-Quintupel weitere 3D-Datenstrukturen, die direkt aus bekannten und bewährten 2D-Datenstrukturen abgeleitet sind. Dazu gehört zum Beispiel die 3D-Corner-Block-List, der 3D-Schnittbaum und die O-Sequenz. Ein Problem für viele Datenstrukturen sind Zyklen. Ein Zyklus liegt vor, wenn zum Beispiel die Module wie folgt zu platzieren sind: A über B, B rechts von C und C hinter A. Dieser Fall ist in Abbildung 4.16 dargestellt.

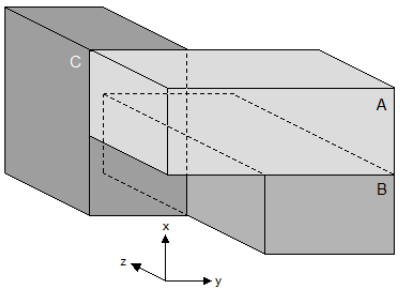

Abb. 4.16: Beispiel für einen Zyklus

Problematisch daran ist in vielen Datenstrukturen, dass sich die Koordinaten für A nur mit den Koordinaten von B, die für B nur mit denen von C und die für C wiederum nur mit den Koordinaten von A berechnen lassen. Das Sequenz-Quintupel hat dieses Problem nicht, da zunächst die x-y-Koordinaten bestimmt werden und danach erst die z-Koordinaten.

Es wurden aber auch neue Datenstrukturen speziell für die 3D-Platzierung entwickelt, wie zum Beispiel Double-Tree-and-Sequence und Labeled-Tree-and-Dual-Sequences. Diese und weitere Datenstrukturen werden in [FL09] bezüglich ihrer Komplexität, der Größe des Lösungsraumes und der möglichen Operationen zum Generieren neuer Lösungen untersucht.

5 Heuristische Optimierungsverfahren für die Platzierung der Superzellen

Um eine geeignete Platzierung der N Superzellen auf k Ebenen zu finden, sollen heuristische Optimierungsverfahren zum Einsatz kommen. Diese wurden ausgewählt, weil kein effizientes exaktes Verfahren bekannt ist.
In dieser Arbeit sollen nur drei heuristische Optimierungsverfahren vorgestellt werden: Simulated Annealing, Threshold Accepting und der Sintflutalgorithmus. Es gibt noch viele andere Verfahren, zum Beispiel genetische Algorithmen, die jedoch in dieser Arbeit nicht untersucht werden, da dies zeitlich nicht realisierbar war.

5.1 Grundlagen

Definition 5.1
Es sei $\mathcal{L}$ die Menge aller zulässigen Lösungen eines kombinatorischen Optimierungsproblems und $f : \mathcal{L} \to \mathbb{R}$ eine Zielfunktion, welche die Güte einer solchen Lösung bestimmt.

Da jede Superzelle in dem 3D-Chip genau einmal zu platzieren ist, sind alle Lösungen zulässig, die jede Superzelle auf genau einer Ebene platzieren. Damit sind die k-Tupel von Sequenz-Paaren, in denen jede Superzelle in genau einem Sequenz-Paar in beiden Sequenzen vorkommt, zulässige Lösungen. Die Zielfunktion, mit der eine solche Platzierung bewertet wird, ist in Kapitel 5.2 beschrieben.

Definition 5.2
Eine **Nachbarschaftsfunktion** $F_N : \mathcal{L} \to \mathcal{P}(\mathcal{L})$ ordnet jeder Lösung $l \in \mathcal{L}$ eine Menge von Lösungen $F_N(l) \subseteq \mathcal{L}$ zu. Diese Menge wird als Nachbarschaft von l bezeichnet. Die Lösungen in der Nachbarschaft von l werden als **Nachbarn** oder **Nachbarschaftslösungen** bezeichnet.

Die Nachbarschaftsfunktion sei in dieser Arbeit definiert durch die Menge aller Lösungen, die aus der aktuellen Lösung durch das Ausführen einer der in Kapitel 4.3.3 beschriebenen Operationen hervorgeht. Die Nachbarschaft einer Lösung ist damit sehr groß: Es gibt allein $\frac{1}{2} \cdot N \cdot (N-1)$ Möglichkeiten, zwei Superzellen zu vertauschen. Auch benötigt die Berechnung des Zielfunktionswerts einer Lösung viel Zeit. Deswegen wäre es sehr zeitaufwändig, die gesamte Nachbarschaft oder eine größere Menge von Nachbarschaftslösungen nach der besten Lösung zu durchsuchen, weshalb Algorithmen, die Derartiges fordern – wie zum Beispiel genetische Algorithmen oder die Tabu-Suche [ACG+99] – hier nicht aufgenommen wurden.

Definition 5.3
Eine Lösung $l^* \in \mathcal{L}$ heißt **globales Optimum** eines Minimierungsproblems, falls gilt: $\forall l \in \mathcal{L} : f(l^*) \leq f(l)$.
Eine Lösung $l^\circ \in \mathcal{L}$ heißt **lokales Optimum**, wenn $\forall l \in F_N(l^\circ) : f(l^\circ) \leq f(l)$.

Ein lokales Optimum kann sehr viel schlechter als das globale Optimum sein. Deswegen ist es wichtig, nicht in lokalen Optima abzubrechen, sondern aus diesen zu entkommen. Um dies zu erreichen, müssen auch schlechtere Lösungen angenommen werden. Die drei hier vorgestellten Algorithmen gehören zu den lokalen Suchalgorithmen und können abhängig von dem jeweiligen Annahmekriterium auch schlechtere Lösungen annehmen.
Grundsätzlich laufen diese lokalen Suchalgorithmen wie folgt ab:
Zunächst wird eine beliebige Startlösung l_0 generiert. Von dieser ausgehend wird in jedem Schritt i eine Nachbarschaftslösung l_N der aktuellen Lösung l_i erzeugt. Wenn die Annahmebedingung des entsprechenden Algorithmus' erfüllt ist, wird $l_{i+1} = l_N$ gesetzt. Andernfalls wird l_i beibehalten. Dieser Vorgang wird solange wiederholt, bis das Abbruchkriterium erfüllt ist.

Algorithmus 5.1.1: Allgemeiner lokaler Suchalgorithmus

Eingabe : Lösungsmenge $\mathcal{L}$
Nachbarschaftsfunktion $F_N : \mathcal{L} \to \mathcal{P}(\mathcal{L})$
Zielfunktion $f : \mathcal{L} \to \mathbb{R}$
Ausgabe : Lösung l_j mit $f(l_j) \leq f(l_i) \quad \forall i \geq 0$
begin
  Erzeuge Startlösung l_0
  $i \leftarrow 0$
  while *Abbruchkriterium ist nicht erfüllt* **do**
    Erzeuge Nachbarlösung $l_N \in F_N(l_i)$
    if *Annahmekriterium ist erfüllt* **then**
      $l_{i+1} \leftarrow l_N$
      $i \leftarrow i + 1$
    else
      $l_{i+1} \leftarrow l_i$
      $i \leftarrow i + 1$

Für das Abbruchkriterium gibt es verschiedene Strategien:

- Der Algorithmus wird nach einer vorgegebenen Anzahl an Iterationsschritten beendet. Damit hat der Nutzer die Möglichkeit, die zeitliche Dauer bis zum Erhalt der Lösung zu steuern.

- Der Algorithmus endet nach einer vorgegebenen Anzahl von Iterationen, in denen kein einziges Mal eine Lösung angenommen wurde, in der Annahme, dass ein lokales Optimum erreicht ist.

- Es ist auch eine Kombination der beiden Abbruchkriterien möglich, um einerseits zu gewährleisten, dass in einer bestimmten Zeit eine Lösung berechnet wird und andererseits unnötig langes Suchen ohne Verbesserung der Lösung vermieden wird.

In dieser Arbeit wurde das erste Abbruchkriterium gewählt. Jedes der drei Verfahren wurde mit drei verschiedenen Iterationslängen getestet.

Es sollte noch angemerkt werden, dass keiner der hier vorgestellten Algorithmen garantieren kann, dass er ein globales Minimum in einer vorgegebenen Zeit findet. Die Annahmekriterien der einzelnen Algorithmen werden im entsprechenden Kapitel erläutert. Da hier ein Minimierungsproblem betrachtet wird, werden diese dann auch für ein Minimierungsproblem dargestellt.

5.2 Die Zielfunktion

Die optimale Lösung sollte den kleinsten Zielfunktionswert haben. Deswegen ist die Gestaltung der Zielfunktion von großer Bedeutung für die Güte der Lösung. Es muss sichergestellt werden, dass alle Optimierungsziele in der Zielfunktion enthalten sind und dass diese Ziele entsprechend ihrer Bedeutung richtig einbezogen werden. Dabei ist zu beachten, dass die optimale Lösung alle Nebenbedingungen erfüllt. Da in dieser Arbeit mit heuristischen Optimierungsverfahren gearbeitet wird, ist es sinnvoll, das Verletzen der Nebenbedingung nur zu bestrafen, statt die Lösung für unzulässig zu erklären. Andernfalls würde der Lösungsraum Lücken aufweisen, die das Lösungsverfahren stark einschränken und den Lösungsbereich möglicherweise in mehrere Teile aufteilen.
In dieser Arbeit liegt das Hauptaugenmerk der Optimierung auf der Minimierung der Fertigungskosten des 3D-Chips. Dabei sind die Zeitbedingungen unbedingt einzuhalten, sonst ist der Chip mit den Geräten, in denen er eingesetzt werden soll, nicht kompatibel und damit unbrauchbar.
Außerdem gibt es noch einige Wünsche an die Gestalt des 3D-Chips. So sollten die Ebenen sich in der Größe nicht zu stark unterscheiden und nicht zu viele TSVs eingebaut werden. Diese beiden Nebenziele werden in der Funktion der Fertigungskosten nicht in dem gewünschten Maße berücksichtigt und werden deswegen durch kleinere Strafen unterstützt.
Die Zielfunktion hat damit folgende Gestalt:

$$f = C + S_{Zeit} + S_{TSV} + S_V.$$

Die Fertigungskosten werden dabei durch den Term C in die Zielfunktion eingebracht, S_{Zeit}, S_{TSV} und S_V sind Strafen. Es ist wichtig, dass die Zielfunktion nicht von den Strafen überreguliert wird – diese sind also so zu wählen, dass sie keinen zu großen Anteil des Zielfunktionswertes ausmachen. Eine Ausnahme bildet dabei die Strafe für das Verletzen der Zeitbedingungen, um zu gewährleisten, dass vorrangig zulässige Lösungen berechnet werden.

5.2.1 Die Fertigungskosten eines Chips

In diesem Abschnitt werden zunächst die Fertigung eines 3D-Chips und die damit einhergehenden Kosten beschrieben.
Ein 3D-Chip wird in drei grundlegenden Schritten gefertigt:

- Fertigung der Ebenen
- Fertigung der TSVs
- Verbinden aller Ebenen

Die Kosten der drei Fertigungsschritte addieren sich auf:

$$C = \sum_{i=1}^{k} C_{Fertigung,i} + \sum_{i=1}^{k} C_{TSV,i} + C_{Verbinden}.$$

Fertigung der Ebenen

Zuerst werden die einzelnen Ebenen gefertigt. Jede Ebene entspricht einem Chip, auf dem nur ein Teil der elektrischen Schaltung aufgebracht wurde. Deswegen entspricht die Herstellung einer einzelnen Ebene dem Fertigungsprozess eines üblichen 2D-Chips [Mös92]:
Die Chipherstellung beginnt mit dem Wafer[7]. Auf dem Wafer werden mehrere hundert identische Chips gleichzeitig gefertigt. Die Anzahl an Chips gegebener Größe A, die auf einen Wafer mit dem Durchmesser D passen, lässt sich zum Beispiel wie folgt abschätzen [DX09]:

$$N(A) = \frac{\pi \cdot D^2}{4 \cdot A} - \frac{\pi \cdot D}{\sqrt{2 \cdot A}}.$$

Die Größe A_i der Ebene i des 3D-Chips berechnet sich folgendermaßen:
Durch die Platzierung der Superzellen auf den einzelnen Ebenen ist für jede Ebene i deren Breite a_i und Höhe b_i durch die Fläche des Floorplans festgelegt. Außerdem ist die Anzahl an TSVs n_i, die auf der Ebene i zu platzieren sind, sowie deren Größe A_{TSV} bekannt. Um die Berechnung der Zielfunktion zu beschleunigen, findet die zeitaufwändige TSV-Platzierung erst am Ende des Algorithmus' statt, zuvor wird die Fläche der TSVs einfach aufaddiert: $A_i = a_i \cdot b_i + n_i \cdot A_{TSV}$.
Dadurch ist aber noch nicht gewährleistet, dass auch alle TSVs platziert werden

[7] Ein Wafer ist eine dünne Scheibe eines Halbleiters, meist Silizium, mit gegebenem Durchmesser.

können. Zwischen zwei TSVs muss ein bestimmter Abstand eingehalten werden, auch dürfen die TSVs nicht zu gleichmäßig und dicht aufgebracht werden, da sie im Prinzip Löcher im Substrat sind und dann wie eine Perforation fungieren würden. Sei d_{TSV} der Abstand, der zwischen zwei TSVs eingehalten werden muss und b_{TSV} die Breite eines TSVs. Nun wird auf der Fläche der Ebene ein Raster mit einer Gitterbreite von $g = d_{TSV} + b_{TSV}$ aufgebracht, so dass auf jedem Rasterpunkt ein TSV platziert werden könnte. Die in dieser Arbeit verwendeten TSVs haben eine Seitenlänge und einen Mindestabstand von jeweils 3µm, damit ergibt sich die Gitterbreite zu 6µm. Wird außerdem gefordert, dass nur $p\%$ der Gitterstücke einen TSV enthalten, so muss für die Fläche der Ebene i gelten: $A_i \geq \frac{g^2}{p} \cdot n_i$.
Darüber hinaus muss gewährleistet sein, dass die darüber liegende Ebene nicht kleiner ist, sonst würden deren Enden in der Luft hängen und sich verbiegen, was den Chip unbrauchbar machen würde. Dieser Sachverhalt wird in Abbildung 5.1 dargestellt.

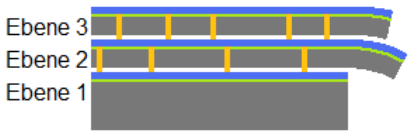

Abb. 5.1: Verbiegung von Ebenen

Da die unterste Ebene als erste Ebene bezeichnet wird, muss gelten: $A_i \geq A_{i+1}$. Ist dies nicht der Fall, so wird $A_i = A_{i+1}$ gesetzt.
Die Ebene i hat damit die Größe $A_i = \max\{a_i \cdot b_i + n_i \cdot A_{TSV},\ \frac{g^2}{p} \cdot n_i,\ A_{i+1}\}$.

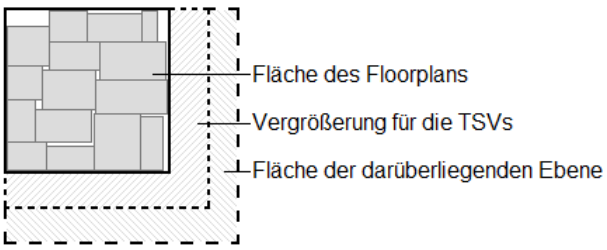

Abb. 5.2: Größe einer Ebene

Abbildung 5.2 zeigt, wie sich die Fläche des Floorplans durch die verschiedenen Rahmenbedingungen vergrößern kann.
Auf dem Wafer werden alle Chips gleichzeitig gefertigt. Die Bauteile und Leitungen werden schichtweise auf den Wafer aufgebracht. Jede einzelne Schicht wird wie folgt erzeugt:

Zuerst wird eine isolierende Schicht aufgebracht, zum Beispiel durch Auftragen einer Oxidationsschicht. Auf dieser wird ein Fotolack aufgetragen und durch eine Maske belichtet. Dadurch entstehen ausgehärtete und weiche Bereiche des Fotolacks. Die unbelichteten, weichen Bereiche können leicht entfernt werden. Bei einem anschließenden Ätzprozess wird in den Gebieten, die nicht von dem ausgehärteten Fotolack geschützt werden, die isolierende Schicht entfernt. Die dabei entstehenden Gräben können nun mit leitfähigem Material gefüllt werden und bilden damit die Bauteile bzw. Leitungen. Damit ist diese Schicht abgeschlossen und die darüber liegende Schicht kann aufgebracht werden.
Bei der Herstellung können verschiedenste Fehler auftreten. Der Anteil an funktionsfähigen Chips mit der Fläche A lässt sich abschätzen durch das sogenannte Negative Binomialmodell [Mis00]:

$$Y_H(A) = (1 + A \cdot \frac{D_0}{\alpha})^{-\alpha}.$$

Dabei ist $D_0 = 0.58\frac{1}{\text{cm}^2}$ die Defektdichte pro Fläche und $\alpha = 3.99$ ein Parameter. Beide Werte wurden in [Mis00] empirisch ermittelt.
Die Kosten C_H des Herstellungsprozesses inklusive der Testkosten von N_{Chip} Chips auf einem Wafer betragen insgesamt ungefähr 1500\$. Zu diesen Waferkosten müssen noch die Maskenkosten addiert werden. Die Masken werden einmal für alle N_{Chip} zu fertigenden Chips hergestellt, die Maskenkosten eines einzelnen Chips ergeben sich damit zu $C_{Maske,Chip} = \frac{C_{Maske}}{N_{Chip}}$.
Die Gesamtkosten zur Herstellung eines Chips der Größe A_i mit n_i TSVs lassen sich dann durch folgende Formel berechnen:

$$C_{Fertigung,i} = \frac{C_H}{N(A_i) \cdot Y_H(A_i)} + \frac{C_{Maske,Chip}}{Y_H(A_i)}.$$

Fertigen der TSVs

Die TSVs werden alle zeitgleich in die Waferscheibe geätzt und anschließend mit einem leitfähigen Material gefüllt. Deswegen sind die TSV-Kosten unabhängig von der Anzahl an TSVs. Für einen Wafer mit einem Durchmesser von 200mm betragen die Kosten C_{TSV} laut [Lau10] zur Zeit 200\$.
Auch bei der Herstellung der TSVs können unbrauchbare Chips entstehen. Mit

welcher Wahrscheinlichkeit und Verteilung Fehler beim Ätzen und Füllen der TSVs auftreten, ist bisher kaum untersucht bzw. publiziert worden. Deswegen wird für jeden TSV eine kleine Fehlerwahrscheinlichkeit P_{TSV} angenommen. Der Anteil an funktionsfähigen Chips mit N_{TSV} TSVs beträgt dann $Y_{TSV}(N_{TSV}) = (1-P_{TSV})^{N_{TSV}}$. Da das Fertigen der TSVs auf jeder Ebene unmittelbar vor oder nach dem Auftragen der Bauteile und Leitungen erfolgt, müssen bei $C_{Herstellung,i}$ und $C_{TSV,i}$ beide Ausbeuten – die für die Herstellung der Leitungen und Bauteile Y_H und die für die TSV-Fertigung Y_{TSV} - berücksichtigt werden. Die Fertigungs- und TSV-Kosten einer Ebene der Größe A_i mit n_i TSVs ergeben sich damit zu:

$$C_{Fertigung,i} = \frac{C_H}{N(A_i) \cdot Y_H(A_i) \cdot Y_{TSV}(n_i)} + \frac{C_{Maske,Chip}}{Y_H(A_i) \cdot Y_{TSV}(n_i)}$$

$$C_{TSV,i} = \frac{C_{TSV}}{N(A_i) \cdot Y_H(A_i) \cdot Y_{TSV}(n_i)}.$$

Verbinden der Ebenen

Sind die Chips für alle Ebenen gefertigt, werden diese übereinander gestapelt, die elektrische Verbindung der TSVs mit der darunterliegenden Ebene über eine kleine Lötkugel gewährleistet und die Ebenen miteinander verklebt. Dabei entstehen Kosten, die hier mit $C_{Verbinden}$ bezeichnet sind. Auch während dieses Prozesses kann es zu Fehlern und damit zu unbrauchbaren Chips kommen. Deswegen ist ein Endtest nötig. Der Anteil der Chips, die hier nicht zerstört werden, sei $Y_{Verbinden}$. Bei den Gesamtkosten ist diese Ausfallrate mit zu berücksichtigen.

Die Funktion der Fertigungskosten

Damit lassen sich die Fertigungskosten eines 3D-Chips, bestehend aus k Ebenen der Größe A_i mit n_i TSVs, $i = 1 \ldots k$, mit folgender Formel abschätzen:

$$C = \sum_{i=1}^{k} \frac{C_H + C_{Maske,Chip} \cdot N(A_i) + C_{TSV}}{N(A_i) \cdot Y_H(A_i) \cdot Y_{TSV}(n_i) \cdot Y_{Verbinden}} + \frac{C_{Verbinden} + C_{Endtest}}{Y_{Verbinden}}$$

5.2.2 Die Strafen

Wie bereits erwähnt, gibt es verschiedenste Strafen. Das ist nötig, da sich nicht alle Optimierungsziele in der Kostenfunktion niederschlagen. So werden die TSVs und die Verteilung der Superzellen auf den Ebenen nur über die Größen der Ebenen berücksichtigt, die Zeitbedingungen finden keine Beachtung in der Kostenfunktion.

Strafe für das Verletzen der Zeitbedingung

In Kapitel 1.1 wurde die Bedeutung der Zeitbedingung bereits erläutert. Wenn es einen Pfad in der Schaltung gibt, auf dem ein Signal mehr Zeit benötigt als vorgegeben, so ist der Chip nicht kompatibel und damit unbrauchbar. Es ist also wichtig, die Zeitbedingung unbedingt einzuhalten. Trotzdem werden Lösungen, welche die Zeitbedingungen verletzen, nicht als unzulässig erklärt, sondern nur hart bestraft. Die Strafe sollte so groß sein, dass ausgehend von einer Lösung, die die Zeitbedingung erfüllt, keine Lösung angenommen werden kann, die diese Bedingung nicht einhält. Die exakte Zeitspanne, die ein Signal zwischen je zwei Register-Zellen benötigt, kann auf dem Floorplan nicht ermittelt werden, da viele Informationen aufgrund der Vereinfachungen, wie dem Zusammenfassen von Zellen zu Superzellen, fehlen. Deswegen wird die Zeitdauer nur abgeschätzt. Dazu wird auf dem Hypergraph des elektrischen Schaltkreises ein Timing-Graph erzeugt, auf dem abgeschätzt wird, wie viel Zeit ein Signal höchstens zwischen zwei Register-Zellen benötigt. Dies erfolgt mit dem in Abschnitt 4.3.1 vorgestellten Algorithmus zur Bestimmung der längsten Wege in einem gerichteten azyklischen Graphen. Dabei sind sowohl die Knoten, als auch die Kanten gewichtet. Die Knoten erhalten als Gewicht die Zeitdauer, die ein Signal benötigt, um die zugehörige Zelle zu passieren. Das Kantengewicht repräsentiert die Zeitdauer, die ein Signal benötigt, um die zugehörige Leitung zu durchlaufen. Die Verzögerung t, die durch eine Leitung der Länge L verursacht wird, lässt sich wie folgt abschätzen [HJR09]: $t(L) = \frac{R' \cdot C'}{2} \cdot L^2$ – mit dem Widerstand pro Längeneinheit R' und der Kapazität pro Längeneinheit C'. Die Länge der Leitung zwischen zwei Superzellen wird über den Abstand ihrer Pins in x- und y-Richtung abgeschätzt. Es wird angenommen, dass sich die Pins in der Mitte der zugehörigen Superzelle befinden. Falls die Superzellen auf verschiedenen Ebenen liegen, muss für jeden später einzufügenden TSV die benötigte Zeitdauer, um diesen zu passieren, addiert werden. Damit wird in jedem Iterationsschritt die Zeitdauer t_C berechnet, die ein Signal

höchstens zwischen zwei Register-Zellen benötigt. Liegt diese Zeit über dem vorgegebenen Wert $t_{max,C}$, so ist die Zeitbedingung verletzt und der Zielfunktionswert ist hart zu bestrafen, indem $a \cdot t_C$ (mit $a > 0$) als Strafe addiert wird. Um sicherzustellen, dass auch die detaillierte Platzierung und Verdrahtung des Chips die Zeitbedingung einhält, sollte $t_{max,C}$ möglichst deutlich unterschritten werden. Deswegen wird auch bei Einhaltung der Zeitbedingung eine kleine Strafe $b \cdot t_C$ (mit $0 < b < a$) addiert. Erst wenn $t_{max,C}$ deutlich unterschritten ist (um den vorgegebenen Wert t_{Buffer}), wird die Zeit-Strafe auf Null gesetzt:

$$S_{Zeit} = \begin{cases} a \cdot t_C & \text{, falls } t_C > t_{max,C} \\ b \cdot t_C & \text{, falls } t_{max,C} \geq t_C > t_{max,C} - t_{Buffer} \\ 0 & \text{sonst} \end{cases}.$$

Bei dem hier verwendeten Beispiel liegt $t_{max,C}$ bei 2.5 ns. Um die Zeitbedingung möglichst oft einzuhalten, wurde $a = 1000$ und $b = 100$ gesetzt. Damit wird gewährleistet, dass auch kleine Verschlechterungen von t_C bestraft werden. Außerdem kann so nach dem Unterschreiten von $t_{max,C}$ eine erneute Annahme einer Lösung, welche die Zeitbedingung in der Abschätzung verletzt, ausgeschlossen werden. Um sicherzustellen, dass die Zeitbedingung nicht nur sehr knapp eingehalten wird und nach dem Einfügen der TSVs keine zulässigen Lösungen erreicht werden können, wurde $t_{Buffer} = 0.05$ ns gesetzt.
Die Zielfunktion wird damit stark von der Zeit-Strafe dominiert. Das ist nötig, um viele zulässige Lösungen zu berechnen.

TSV-Strafe

Um eine Minimierung der TSV-Anzahl zu unterstützen, auch wenn theoretisch mehr TSVs auf den Ebenen platziert werden könnten, wird für jeden TSV oberhalb einer maximal tolerierten TSV-Anzahl $N_{max,TSV}$ eine Strafe s addiert:

$$S_{TSV}(N_{TSV}) = \begin{cases} 0 & \text{, falls } N_{TSV} \leq N_{max,TSV} \\ (N_{TSV} - N_{max,TSV}) \cdot s & \text{sonst} \end{cases}.$$

Diese Strafe wurde eingeführt, da bisher noch keine genauen Erkenntnisse über die TSVs bekannt sind. Deswegen sollten zunächst so wenig TSVs wie möglich eingesetzt werden.

Strafe für ungünstige Verteilung der Superzellen

Abschließend wird noch eine Strafe für ungünstige Verteilung der Superzellen eingesetzt. Damit sind Platzierungen gemeint, die sehr kleine Ebenen haben, auf denen lediglich einige wenige Superzellen platziert wurden. Um eine solche Verteilung der Superzellen auf den Ebenen zu vermeiden, wurde folgende Strafe eingeführt:
Sei N_{min} die minimale und N_{max} die maximale Anzahl an Superzellen auf einer Ebene. Dann gilt:

$$S_V = \begin{cases} 0 & \text{, falls } \frac{N_{max}}{N_{min}} < \Delta_B \\ S(N_{max}, N_{min}) & \text{sonst} \end{cases}.$$

Die Strafe S kann unterschiedlich berechnet werden, je nachdem, wie stark eine ungleichmäßige Verteilung der Superzellen auf den Ebenen bestraft werden soll.
In dieser Arbeit wurde $\Delta_B = 2$ und $S(N_{max}, N_{min}) = 10 \cdot (N_{max} - N_{min})$ gesetzt. Damit kann praktisch ausgeschlossen werden, dass sehr kleine Ebenen entstehen, da die Strafe bei $\frac{N_{max}}{N_{min}} \geq 2$ hinreichend groß ist. Das führt dazu, dass eine solche Lösung sofort abgelehnt wird.

5.3 Simulated Annealing

5.3.1 Der Algorithmus

Simulated Annealing [AK90, Aze92] ist ein in der Praxis häufig eingesetztes Verfahren zum approximativen Lösen kombinatorischer Optimierungsprobleme. Diese Methode wurde 1983 von Kirkpatrick, Gelatt und Vecchi, sowie unabhängig davon 1985 von Černy, entwickelt. Grundlage dafür waren der Metropolisalgorithmus [Aze92] und der natürliche Prozess des Abkühlens.
Die Lösungsstrategie ist einfach: Beginnend mit einer Startlösung wird in jedem Schritt ausgehend von der aktuellen Lösung l_i eine Nachbarschaftslösung $l_N \in F_N(l_i)$ generiert. Diese Lösung wird – abhängig von der Güte der Lösungen l_i und l_N, sowie der Temperatur T_i – mit einer bestimmten Wahrscheinlichkeit angenommen oder abgelehnt.
Sei $T_i \geq 0$, $i = 0, 1, 2 \ldots$ eine monoton fallende Folge von Temperaturen, dass heißt es

gilt: $\forall i \geq 0 : T_i \geq T_{i+1}$. Die Nachbarschaftslösung wird mit Sicherheit angenommen, wenn sie besser als die aktuelle Lösung ist, andernfalls nur mit der Wahrscheinlichkeit $P = e^{\frac{f(l_i)-f(l_N)}{T}}$. Die Wahrscheinlichkeit, die Nachbarschaftslösung l_N anzunehmen, ist damit:

$$P(l_{i+1} \leftarrow l_N \in F_N(l_i)) = \min\{1, e^{\frac{f(l_i)-f(l_N)}{T_i}}\}.$$

Die Annahmewahrscheinlichkeit einer Nachbarlösung l_N ausgehend von der Lösung l_i mit $f(l_i) = 100$ wurde in Abbildung 5.3 dargestellt. Die Kurve der Annahmewahrscheinlichkeit wurde für drei Temperaturen eingezeichnet. In der Abbildung ist zu sehen, dass niedrige Temperaturen dazu führen, dass Verschlechterungen mit einer kleineren Wahrscheinlichkeit angenommen werden und dass bei sehr hohen Temperaturen die Annahmewahrscheinlichkeit aller Nachbarschaftslösungen steigt.

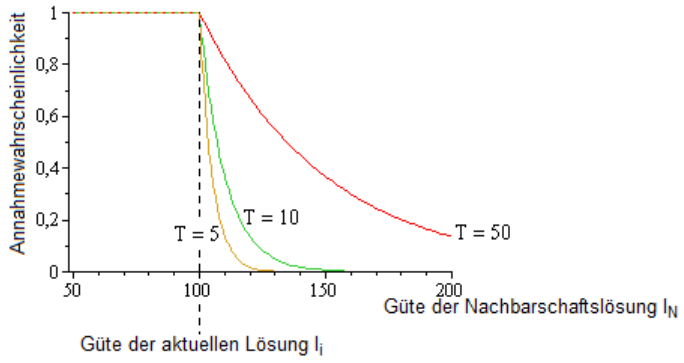

Abb. 5.3: Annahmewahrscheinlichkeit von Nachbarlösungen abhängig von deren Güte und der Temperatur

Für die Reihe der Temperaturen gibt es verschiedene Ansätze. Bei allen wird mit einer hohen Temperatur begonnen – um dem Algorithmus am Anfang viel Spielraum zu lassen – und diese dann langsam abgesenkt. Dabei müssen die Temperaturen auf das eigene Problem abgestimmt werden, um möglichst gute Ergebnisse zu erhalten. Werden die Temperaturen zu groß gewählt, so werden schlechte Lösungen zu schnell angenommen und der Lösungsraum nahezu ungesteuert durchsucht. Sind sie hingegen zu klein oder sinken zu schnell ab, werden nur wenige Lösungen akzeptiert. Damit können lokale Minima nur schwer überwunden werden.

5.3.2 Ergebnisse

Der Algorithmus Simulated Annealing wurde an einer Beispiel-Schaltung getestet. Es wurde untersucht, welche Ergebnisse mit verschiedenen Abkühlungskurven erzielt wer-

den. Dabei stellt sich insbesondere die Frage, wie stabil die Lösungen sind – also wie stark sich die Lösungen verschiedener Versuche voneinander unterscheiden. Werden in mehreren Durchgängen immer Lösungen mit einem ähnlichen Zielfunktionswert errechnet, so ist das Verfahren stabil – schwanken die Ergebnisse dagegen stark, so ist es instabil.

In der Praxis wird oft vorgeschlagen, die Temperatur in jedem Berechnungsschritt zu verringern. In dieser Arbeit wurde jedoch eine Liste von Temperaturen $[T_i, i = 1, 2, \ldots 50]$ erzeugt, um mit jeder Temperatur eine vorgegebene Anzahl an Iterationsschritten N_{It} zu durchlaufen. Das hat den Vorteil, dass durch einfaches Verändern von N_{It} die Länge des Algorithmus' gesteuert werden kann, ohne dass die Temperaturen angepasst werden müssen.

Es wurden drei Abkühlungskurven mit verschiedenen Startwerten T_0 untersucht, deren Temperaturen $T_i,\ i = 1, 2, \ldots 45$ wie folgt berechnet wurden:

- linear: $T_i = T_0 \cdot (1 - \frac{i-1}{45})$
- exponentiell: $T_i = T_0 \cdot 0.9^{i-1}$
- logarithmisch: $T_i = \frac{T_0}{ln(i+1)} - \frac{i}{20}$

Die letzten fünf Temperaturen $T_{46}, \ldots, T_{50}$ wurden auf Null gesetzt, um am Ende möglichst ein lokales Minimum zu erreichen.

Die daraus resultierenden Temperaturverläufe für $T_0 = 15$ sind in Abbildung 5.4 dargestellt.

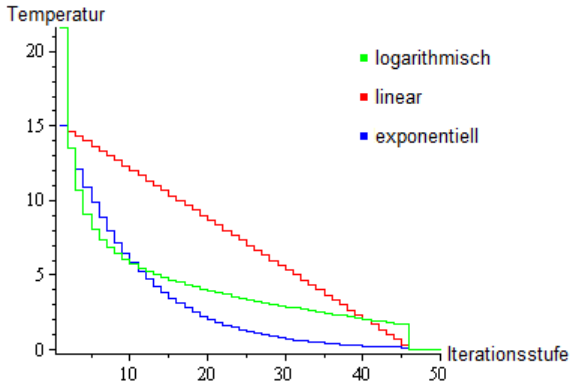

Abb. 5.4: Abkühlungskurven für die Temperaturen

Linearer Temperaturverlauf

Das lineare Abkühlen der Temperaturen wurde mit verschiedenen Startwerten vollzogen. Die folgende Tabelle enthält die Ergebnisse dieser Methode. Dabei ist H_A die Häufigkeit, mit der Nachbarschaftslösungen während des Algorithmus' angenommen wurden und H_{Time} die Häufigkeit, mit der die Endlösung nach Einfügen der TSVs die Zeitbedingung einhält. Die Überschreitung der Zeitbedingung ist $t_\Delta = t_C - t_{max,C}$ (falls $t_C > t_{max,C}$). Die durchschnittliche TSV-Anzahl wird mit N_{TSV} bezeichnet. Es wurde mit 1000 und 2000 Iterationen pro Temperatur 100-mal gerechnet.

T_0	H_A	Zielfunktionswert			Kosten	$\varnothing N_{TSV}$	H_{Time}	t_Δ
	∅ in %	∅	Min	Max	∅ in ct		in %	in ns
$N_{It} = 1000$								
5	20	439	264	2795	16.07	1716	9	0.072
10	27	332	264	508	15.66	1281	27	0.049
15	32	354	263	2610	15.65	1278	30	0.053
20	37	418	264	2790	15.66	1256	36	0.054
25	41	535	264	2864	15.84	1396	14	0.060
30	44	762	264	2888	15.88	1396	18	0.066
$N_{It} = 2000$								
5	20	386	262	578	15.84	1561	16	0.062
10	27	302	262	411	15.40	1124	56	0.040
15	32	293	262	388	15.36	1075	66	0.037
20	37	307	262	474	15.43	1147	51	0.041
25	41	314	262	507	15.46	1189	51	0.047
30	44	319	262	513	15.51	1212	44	0.045

Tab. 5.1: Ergebnisse mit Simulated Annealing bei linear sinkenden Temperaturen aus 100 Versuchen

Der Tabelle 5.1 ist zu entnehmen, dass die durchschnittliche Annahmehäufigkeit H_A unabhängig von der Anzahl an Iterationen ist. Wie zu erwarten, steigt H_A mit der Temperatur an.

Beim Vergleich der erhaltenen Zielfunktionswerte ist festzustellen, dass die minimalen Werte bei allen Varianten zwischen 262 und 264 liegen. Bei der größeren Anzahl an Iterationen wurden geringfügig bessere minimale Zielfunktionswerte ermittelt. Der Zeit-Buffer wurde allerdings in keinem der 100 Versuche unterschritten, sonst würden die minimalen Zielfunktionswerte deutlich unter 245 liegen. Der größte berechnete Zielfunktionswert bei $N_{It} = 1000$ liegt bei fast allen Startwerten weit über 2500, da in der Abschätzung der innerhalb des Chips benötigten Zeit (ohne TSVs) die Zeitbedingung verletzt wurde. Lediglich bei $T_0 = 15$ wurde diese wichtige Nebenbedingung in dem Chip ohne TSVs immer eingehalten. Deswegen liegt der größte erhaltene Zielfunktionswert deutlich unter 2500.

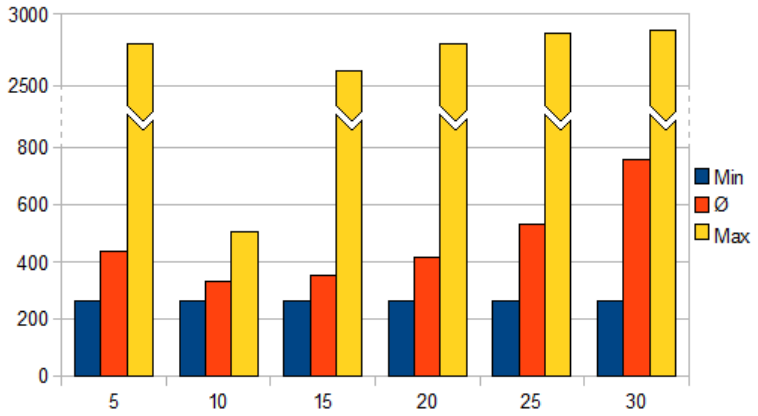

Abb. 5.5: Vergleich der berechneten Zielfunktionswerte mit Simulated Annealing, $N_{It} = 1000$ und linearem Temperaturverlauf

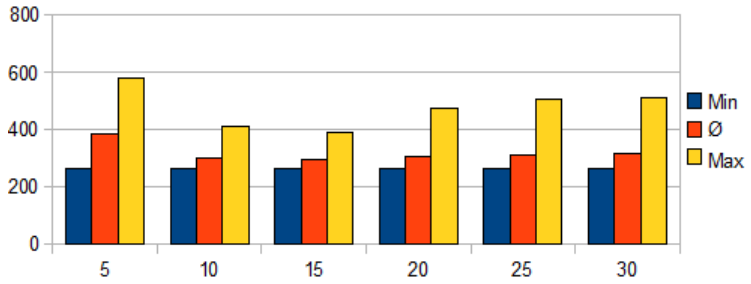

Abb. 5.6: Vergleich der berechneten Zielfunktionswerte mit Simulated Annealing, $N_{It} = 2000$ und linearem Temperaturverlauf

In den Abbildungen 5.5 und 5.6 wurden die Zielfunktionswerte graphisch dargestellt. Auf der Ordinate sind die erhaltenen Zielfunktionswerte abgetragen, auf der Abszisse die Startwerte T_0. Für jeden Startwert ist ein Balken für den minimalen, den durchschnittlichen und den maximalen Zielfunktionswert aus 100 Versuchen eingezeichnet. Die besten durchschnittlichen Zielfunktionswerte wurden bei $N_{It} = 1000$ mit $T_0 = 10$ berechnet. Mit größer werdender Starttemperatur steigen bei 1000 Iterationen pro Temperatur auch die durchschnittlich berechneten Zielfunktionswerte, da bei sehr hohen Temperaturen schlechtere Lösungen mit einer höheren Wahrscheinlichkeit angenommen werden. Das kann dazu führen, dass sich zu Beginn des Algorithmus' die Lösung nur wenig verbessert. Auch bei kleinerem T_0 werden im Durchschnitt schlechtere Lösungen berechnet. Das liegt daran, dass das Vertauschen zweier Superzellen verschiedener Ebenen und das Verschieben einer Superzelle auf eine andere Ebene nur sehr selten angenommen werden, da diese Operationen meist eine starke Veränderung des Floorplans hervorrufen. Deswegen verschlechtert sich der Zielfunktionswert nach einer solchen Operation häufig stärker als nach dem Verändern der Platzierung auf

nur einer Ebene.

Bei 2000 Iterationen pro Temperatur werden diese Effekte nicht so deutlich. Das liegt vermutlich daran, dass die Iterationsanzahl hinreichend groß ist. Der Algorithmus hat dann auch bei zu kleinen Temperaturen hinreichend Zeit, um mit kleinen Veränderungen lokale Minima zu überwinden. Auch zu große Temperaturen können gut ausgeglichen werden, da diese sinken und hinreichend viele Iterationen möglich sind, um mit den folgenden, kleineren Temperaturen eine gute Lösung zu berechnen.

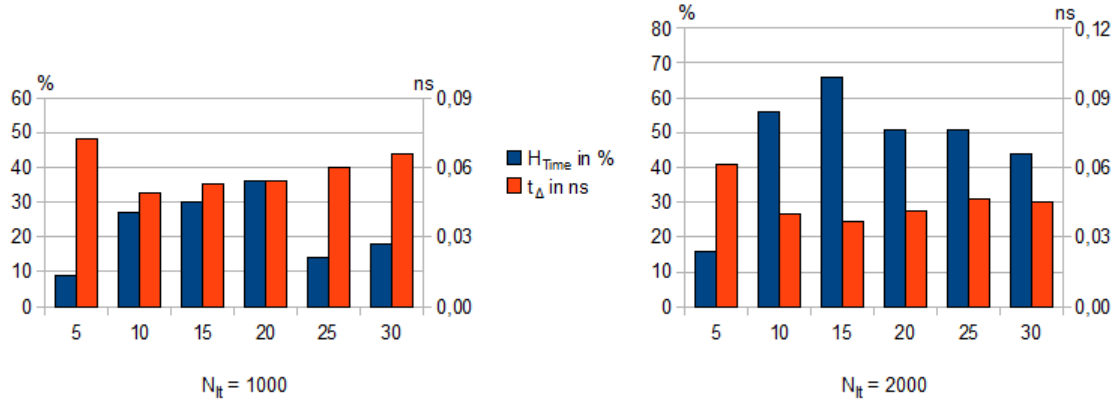

Abb. 5.7: Einhaltung der Zeitbedingung und durchschnittliche Überschreitung von $t_{max,C}$ mit Simulated Annealing und linearem Temperaturverlauf

In Abbildung 5.7 sind zwei Diagramme dargestellt, in denen die Häufigkeit, mit der die Endlösung nach dem Einfügen der TSVs die Zeitbedingung einhält und die durchschnittliche Überschreitung von $t_{max,C}$ eingetragen sind. Die linke Graphik zeigt dabei die Ergebnisse mit 1000 Iterationen pro Temperatur, in dem rechten Diagramm werden die Ergebnisse für $N_{It} = 2000$ dargestellt.

Auf den ersten Blick fällt auf, dass mit 2000 Iterationen pro Temperatur die Zeitbedingung deutlich öfter eingehalten wurde als mit $N_{It} = 1000$. Auch die durchschnittliche Überschreitung der maximal zulässigen Zeit, die ein Signal innerhalb des Chips benötigen darf, ist deutlich niedriger. Es ist zu erkennen, dass die Zeitbedingung bei $N_{It} = 1000$ mit $T_0 = 20$ am häufigsten eingehalten wird – immerhin 36-mal. Bei doppelter Anzahl an Iterationen wurde die Zeitbedingung 66-mal mit $T_0 = 15$ eingehalten. Auch bei den Startwerten 10, 20 und 25 ist mehr als die Hälfte der berechneten Lösungen auch nach dem Einfügen der TSVs noch zulässig.

In den beiden Diagrammen aus Abbildung 5.8 sind die durchschnittlichen Kosten dargestellt. Auf der Ordinate sind die Kosten abgetragen, auf der Abszisse die Startwerte. Das linke Diagramm zeigt die Ergebnisse mit 1000 Iterationen pro Temperatur,

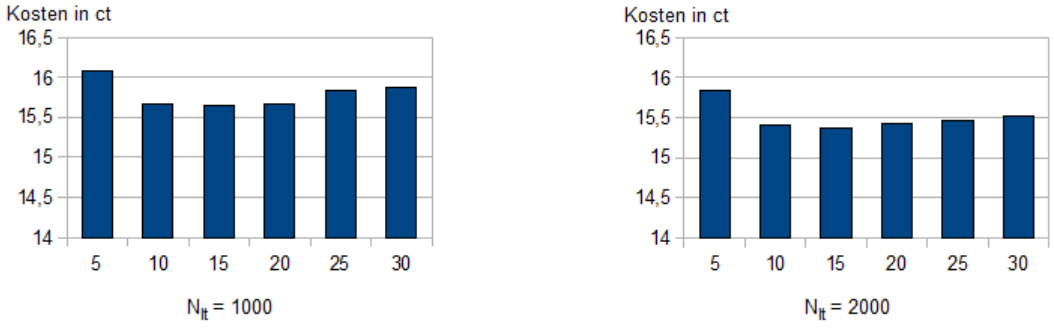

Abb. 5.8: Vergleich der durchschnittlichen Kosten pro Chip mit Simulated Annealing und linearem Temperaturverlauf

das rechte die durchschnittlichen Kosten bei $N_{It} = 2000$.

Es ist zu sehen, dass die durchschnittlichen Kosten pro Chip bei $N_{It} = 2000$ insgesamt kleiner sind als bei $N_{It} = 1000$. Die besten Werte wurden mit Kosten von 15.65 ct bzw. 15.36 ct bei 1000 und 2000 Iterationen pro Temperatur mit dem Startwert 15 ermittelt.

Abschließend kann gesagt werden, dass beim Simulated Annealing mit linearem Absenken der Temperatur die besten Ergebnisse mit $T_0 = 15$ und $N_{It} = 2000$ berechnet wurden. Bei 1000 Iterationen pro Temperatur war das Verfahren bei $T_0 = 10$ am stabilsten, deswegen wurden dort im Schnitt die besten Zielfunktionswerte erzielt. Die Zeitbedingung wurde allerdings mit $T_0 = 20$ am häufigsten eingehalten.

Exponentielles Absenken der Temperatur

Im Folgenden wird dargestellt, welche Ergebnisse mit Simulated Annealing und dem exponentiellen Absenken der Temperatur ermittelt wurden. Dabei wurde wiederum mit verschiedenen Starttemperaturen T_0 gerechnet.

Die Ergebnisse werden zunächst tabellarisch dargestellt. In der Tabelle ist H_A die Häufigkeit, mit der Lösungen angenommen wurden. Daneben sind die Zielfunktionswerte, die Kosten, die TSV-Anzahl N_{TSV}, die Häufigkeit H_{Time}, mit der die Zeitbedingung nach dem Einfügen der TSVs eingehalten wird und mit t_Δ die Überschreitung der maximal zulässigen Zeit $t_{max,C}$ eingetragen.

Der Tabelle 5.2 ist zu entnehmen, dass die durchschnittliche Annahmehäufigkeit nicht von der Länge des Algorithmus' abhängig ist.

Bei 1000 Iterationen pro Temperatur und Startwerten ab 45 schwanken die Lösungen

T_0	H_A	Zielfunktionswert			Kosten	$\varnothing N_{TSV}$	H_{Time}	t_Δ
	$\varnothing$ in %	$\varnothing$	Min	Max	$\varnothing$ in ct		in %	in ns
$N_{It} = 1000$								
5	14	482	265	641	16.34	2040	1	0.090
15	21	382	263	543	15.84	1556	9	0.057
25	25	370	263	570	15.81	1476	16	0.061
35	29	365	263	577	15.78	1459	16	0.057
45	32	402	264	2818	15.87	1527	15	0.062
65	36	444	264	2891	15.89	1517	14	0.060
85	40	379	264	2651	15.79	1407	20	0.058
105	43	642	264	2826	15.93	1482	11	0.061
125	44	614	264	2880	15.92	1451	18	0.069
$N_{It} = 2000$								
5	14	437	260	600	16.05	1826	5	0.072
15	21	334	261	500	15.51	1299	30	0.045
25	26	324	261	511	15.48	1241	40	0.044
35	29	332	261	516	15.56	1292	32	0.043
45	32	329	262	527	15.79	1269	37	0.046
65	36	332	262	503	15.56	1287	36	0.046
85	40	326	262	564	15.51	1253	42	0.044
105	43	328	262	493	15.56	1271	32	0.043
125	45	348	261	2772	15.56	1248	40	0.050

Tab. 5.2: Ergebnisse mit Simulated Annealing und exponentiell sinkenden Temperaturen aus 100 Versuchen

stark, da die Zeitbedingung in der Abschätzung ohne TSVs häufig verletzt wird. Allerdings sind die zulässigen Lösungen häufig besser als bei den niedrigeren Starttemperaturen, sonst würde die Zeitbedingung in der Endlösung häufiger verletzt werden. Der kleinste durchschnittliche Zielfunktionswert liegt bei 365 und wurde mit $T_0 = 35$ ermittelt.

Die Verdoppelung der Iterationsanzahl stabilisiert das Verfahren deutlich. Die Zeitbedingung wird in der Abschätzung lediglich bei $T_0 = 125$ verletzt. Insgesamt sind die Ergebnisse bei $N_{It} = 2000$ deutlich besser, da sowohl die Zielfunktionswerte und Kosten kleiner sind, als auch die Zeitbedingung häufiger eingehalten wurde. Der minimale durchschnittliche Zielfunktionswert wurde mit $T_0 = 25$ berechnet und beträgt 324. Allerdings wurden mit den Startwerten 45, 85 und 105 ebenfalls durchschnittliche Zielfunktionswerte unter 330 ermittelt. Diese Tatsache lässt den Schluss zu, dass diese Startwerte ebenfalls gut sind.

In den Abbildungen 5.9 und 5.10 sind die berechneten Zielfunktionswerte graphisch dargestellt. Auf der Abszisse sind die verschiedenen Startwerte abgetragen, auf der Ordinate die ermittelten Zielfunktionswerte. Dabei sind jeweils der beste, der durchschnittliche und der schlechteste Zielfunktionswert eingezeichnet.

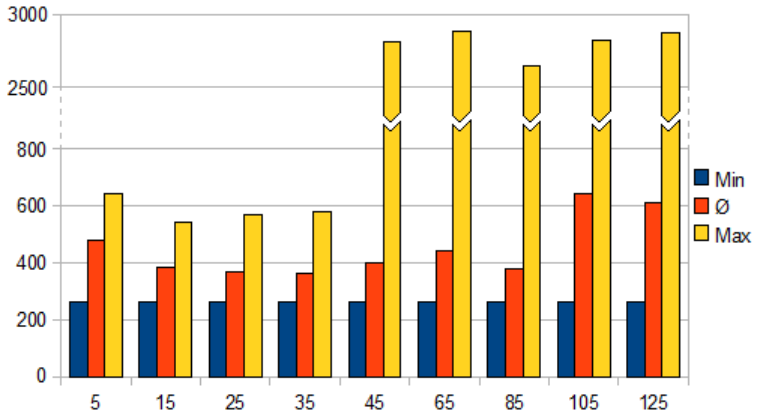

Abb. 5.9: Vergleich der berechneten Zielfunktionswerte mit Simulated Annealing, $N_{It} = 1000$ und exponentiellem Temperaturverlauf

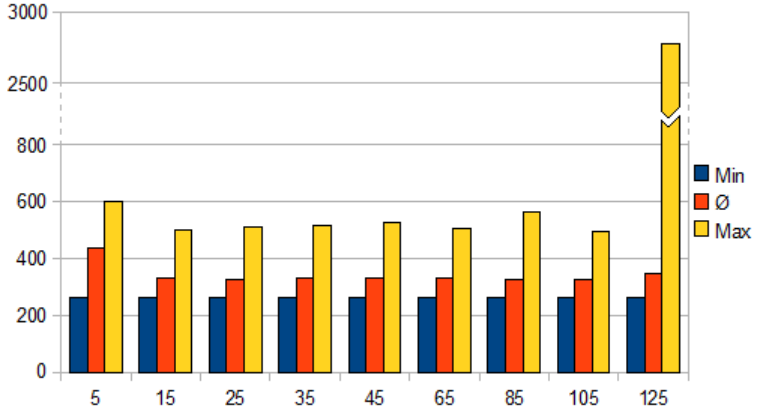

Abb. 5.10: Vergleich der berechneten Zielfunktionswerte mit Simulated Annealing, $N_{It} = 2000$ und exponentiellem Temperaturverlauf

Den Diagrammen ist zu entnehmen, dass der kleinste berechnete Zielfunktionswert bei 1000 und 2000 Iterationen pro Temperatur fast konstant ist. Bei $N_{It} = 1000$ ist zu erkennen, dass bei großen Starttemperaturen das Verfahren weniger stabil ist, da dort der größte Zielfunktionswert deutlich höher ist. Deswegen steigt auch der durchschnittliche Zielfunktionswert. Die besten Werte wurden bei $N_{It} = 1000$ mit $T_0 = 35$ berechnet.

Bei 2000 Iterationen pro Temperatur wurden bei allen Starttemperaturen ungefähr die gleichen Werte berechnet, lediglich bei $T_0 = 5$ wurde ein höherer durchschnittlicher Zielfunktionswert ermittelt. Der größte erhaltene Zielfunktionswert wurde mit $T_0 = 125$ berechnet. Da aber bei diesem Startwert der durchschnittliche Zielfunktionswert kaum über denen der anderen Startwerte liegt, ist dieser stark abweichende Wert vermutlich nur ein Ausreißer.

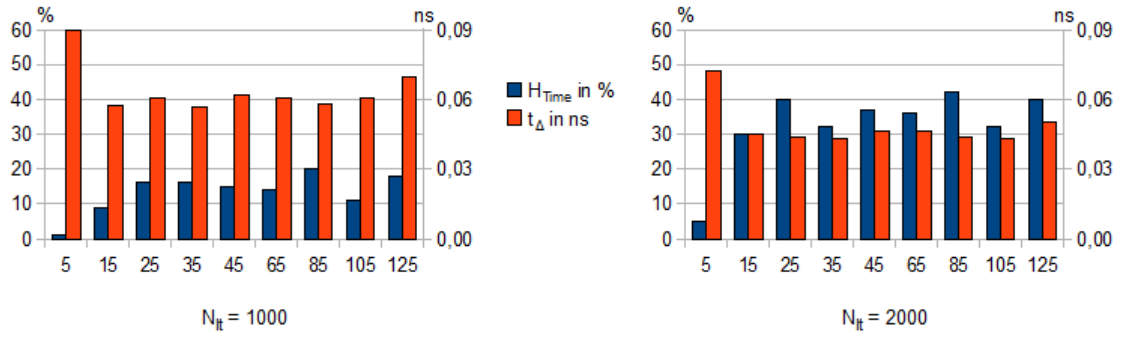

Abb. 5.11: Einhaltung der Zeitbedingung und durchschnittliche Überschreitung von $t_{max,C}$ mit Simulated Annealing und exponentiellem Temperaturverlauf

Wie der Tabelle bereits zu entnehmen war, sind nicht nur die Zielfunktionswerte, sondern auch das Einhalten der Zeitbedingung und die durchschnittliche Überschreitung von $t_{max,C}$ fast auf der gesamten Bandbreite an Startwerten nahezu konstant. Diese Tatsache wird vor allem in den Diagrammen aus Abbildung 5.11 deutlich. Darin sind die Häufigkeit, mit der die Zeitbedingung nach dem Einfügen der TSVs eingehalten wurde und die durchschnittliche Überschreitung von $t_{max,C}$ bei Verletzung der Zeitbedingung eingetragen. Es ist zu sehen, dass sowohl bei 1000, als auch bei 2000 Iterationen pro Temperatur die wenigsten zulässigen Lösungen mit $T_0 = 5$ ermittelt wurden. Auch die durchschnittliche Überschreitung der maximal zulässigen Zeit ist hier deutlich größer. Mit den anderen Startwerten wurden recht ähnliche Ergebnisse erzielt. Bei $N_{It} = 1000$ wurden mit den Starttemperaturen 25 bis 125 im Schnitt 15 zulässige Lösungen ermittelt, bei 2000 Iterationen pro Temperatur ergaben sich mit den Temperaturen 15 bis 125 im Schnitt 36 zulässige Lösungen. Die Schwankungen von H_{Time} haben ihre Ursache vermutlich in der recht kleinen Anzahl an Versuchen.

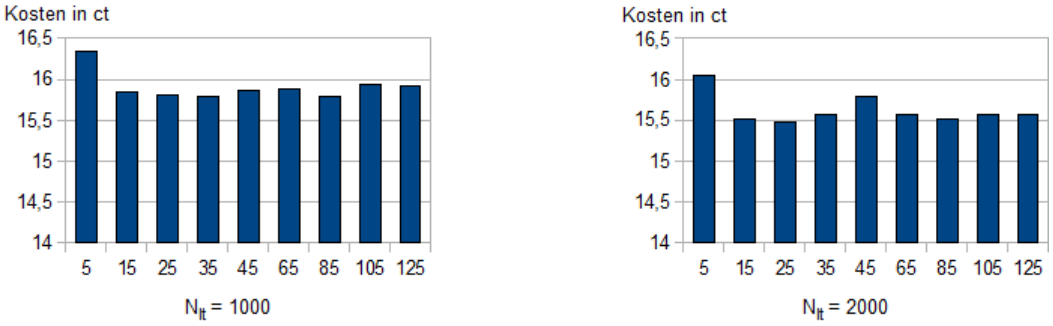

Abb. 5.12: Vergleich der durchschnittlichen Kosten pro Chip mit Simulated Annealing und exponentiellem Temperaturverlauf

In Abbildung 5.12 sind zwei Diagramm für die Kosten pro Chip dargestellt. Dabei sind auf der Ordinate die Kosten abgetragen, auf der Abszisse die Startwerte. Das linke Diagramm stellt die Ergebnisse für 1000 Iterationen pro Temperatur dar, das rechte die durchschnittlichen Kosten bei $N_{It} = 2000$. Es ist wiederum gut zu sehen, dass bei fast allen Startwerten sehr ähnliche Ergebnisse ermittelt wurden. Lediglich bei $T_0 = 5$ liegen deutlich höhere Kosten vor. Insgesamt fallen die Kosten pro Chip bei doppelter Iterationszeit deutlich niedriger aus.
Abschließend ist anzumerken, dass beim exponentiellen Absenken der Temperatur mit einer großen Bandbreite von Startwerten gute Ergebnisse erzielt wurden. Deswegen kann mit dieser Methode zum Absenken der Temperatur gut nach Startwerten bei einem neuen, unbekannten Problem gesucht werden, da eine größere Schrittweite zwischen den Starttemperaturen verschiedener Versuche genutzt werden kann.

Logarithmisches Absenken der Temperatur

Diese Strategie für das Absenken der Temperatur wurde ausgewählt, da sie häufig in der Literatur erwähnt wird. Außerdem wurde deutlich, dass beim exponentiellen Absenken der Temperatur schon früh sehr kleine Temperaturen und damit kleine Annahmewahrscheinlichkeiten vorliegen. Bei einem logarithmischen Absenken sind die Temperaturen im Schnitt deutlich höher, lediglich am Anfang des Algorithmus' sinken sie schnell ab. Da eine rein logarithmische Funktion $T_i = \frac{T_0}{ln(i+1)}$ mit $i \geq 15$ kaum fällt, wurde der lineare Anteil eingebracht. Damit sinken die Temperaturen bei den letzten Iterationsstufen deutlich stärker.

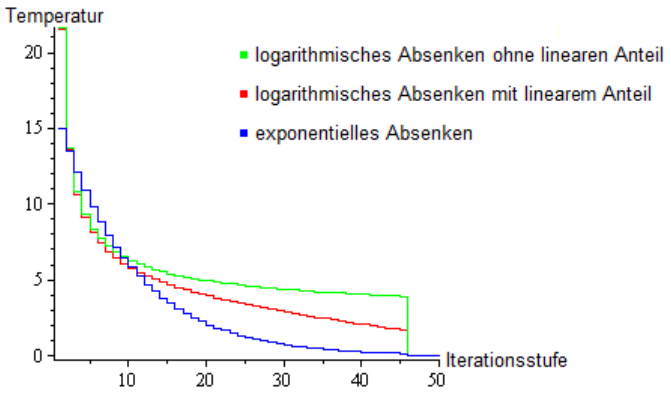

Abb. 5.13: Vergleich der Kurvenverläufe bei exponentiellem und logarithmischem Absenken der Temperatur mit und ohne linearen Anteil

In Abbildung 5.13 sind die Temperaturverläufe für das logarithmische Absenken der Temperaturen mit und ohne linearen Anteil und für das exponentielle Absenken dargestellt. Es ist deutlich zu sehen, dass ohne den linearen Anteil die Temperaturen T_i, $i \geq 15$ deutlich größer sind. Damit können am Ende des Algorithmus' zu viele Lösungen angenommen werden, so dass dort kaum optimiert wird. Auch wird deutlich, dass die letzten Temperaturen beim logarithmischen Absenken deutlich über denen beim exponentiellen Absenken liegen. Beim logarithmischen Senken der Temperatur können also am Ende mehr Lösungen angenommen werden.

T_0	H_A	Zielfunktionswert			Kosten	$\varnothing N_{TSV}$	H_{Time}	t_Δ
	$\varnothing$ in %	$\varnothing$	Min	Max	$\varnothing$ in ct		in %	in ns
$N_{It} = 1000$								
10	22	416	263	2708	15.98	1607	7	0.064
15	27	659	264	2763	15.87	1335	17	0.057
20	31	1058	266	2779	15.92	1286	4	0.055
25	35	941	265	2690	15.79	1149	7	0.042
30	38	1371	266	2663	15.80	1140	5	0.045
35	41	1654	267	2757	15.90	1193	0	0.053
$N_{It} = 2000$								
10	22	355	262	568	15.70	1415	21	0.050
15	27	342	262	2781	15.57	1219	41	0.048
20	32	296	262	386	15.44	1098	56	0.045
25	35	355	263	2633	15.42	1039	46	0.028
30	38	367	263	2638	15.34	954	46	0.016
35	41	538	264	2660	15.51	1031	34	0.031

Tab. 5.3: Ergebnisse mit Simulated Annealing und logarithmisch sinkenden Temperaturen (mit linearem Anteil) aus 100 Versuchen

In Tabelle 5.3 werden die Ergebnisse der Berechnungen mit logarithmisch sinkenden Temperaturen gezeigt. Es sind wiederum die durchschnittliche Annahmehäufigkeit H_A, die Zielfunktionswerte, die Kosten, die TSV-Anzahl N_{TSV}, die Häufigkeit H_{Time}, mit der Lösungen die Zeitbedingung einhalten und mit t_Δ die Überschreitung von $t_{max,C}$ eingetragen.
Es ist zu sehen, dass sowohl bei 1000, als auch bei 2000 Iterationen pro Temperatur die berechneten Zielfunktionswerte stark schwanken. Die Zeitbedingung wird in der Abschätzung ohne TSVs auch bei $N_{It} = 2000$ häufig verletzt. Bei 1000 Iterationen pro Temperatur wird diese wichtige Nebenbedingung sogar sehr oft nicht eingehalten, sonst würden die durchschnittlichen Zielfunktionswerte deutlich niedriger ausfallen. Die besten Lösungen wurden bei $N_{It} = 1000$ mit $T_0 = 10$ und bei $N_{It} = 2000$ mit $T_0 = 20$ erzielt.

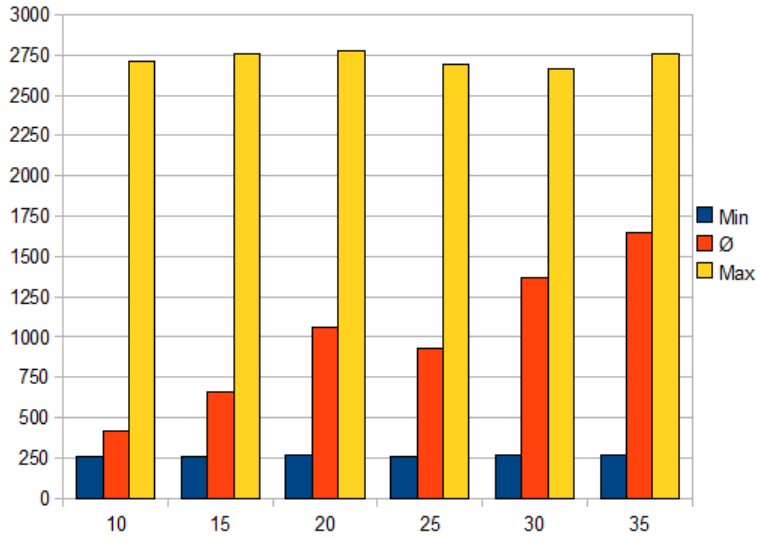

Abb. 5.14: Vergleich der Zielfunktionswerte mit Simulated Annealing, $N_{It} = 1000$ und logarithmischem Absenken der Temperatur

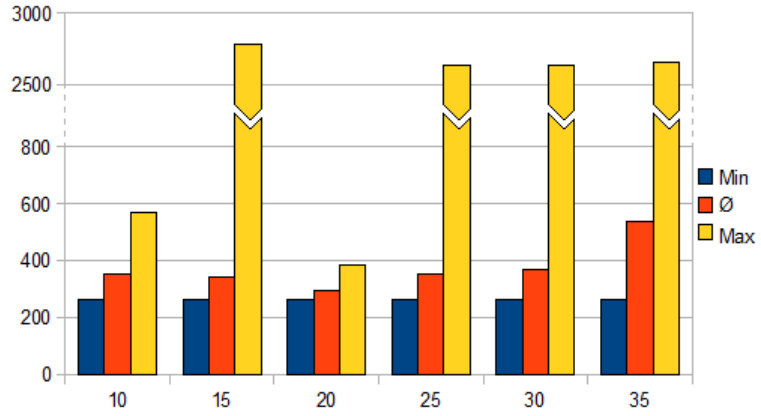

Abb. 5.15: Vergleich der Zielfunktionswerte mit Simulated Annealing, $N_{It} = 2000$ und logarithmischem Absenken der Temperatur

In den Abbildungen 5.14 und 5.15 werden die berechneten Zielfunktionswerte graphisch dargestellt. Auf der Abszisse sind die minimalen, durchschnittlichen und maximalen Zielfunktionswerte abgetragen, auf der Ordinate die Startwerte T_0. Es ist gut zu sehen, dass der durchschnittliche Zielfunktionswert bei $N_{It} = 1000$ und den Startwerten 20 bis 35 sehr groß ist. Das liegt daran, dass in der Abschätzung ohne TSVs die maximale Zeit, die ein Signal innerhalb des Chips verweilen darf, häufig überschritten wurde. Bei 2000 Iterationen pro Temperatur wurde die Zeitbedingung öfter eingehalten. Deswegen sind die Zielfunktionswerte insgesamt kleiner. Trotzdem ist das Verfahren immer noch recht instabil.

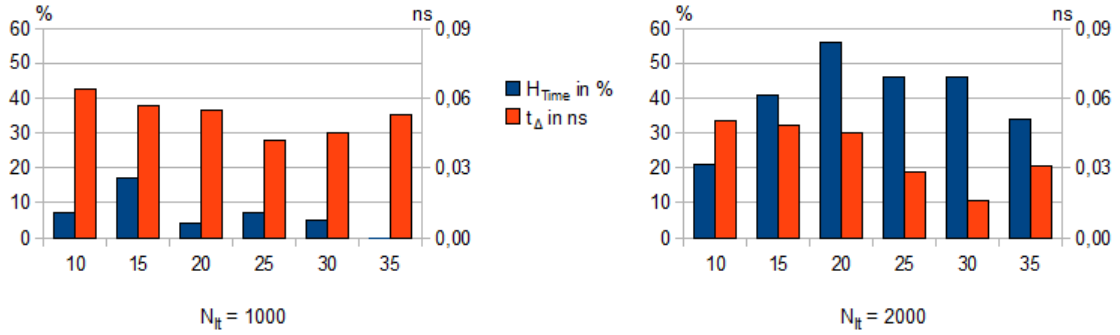

Abb. 5.16: Einhaltung der Zeitbedingung und durchschnittliche Überschreitung von $t_{max,C}$ mit Simulated Annealing und logarithmischem Absenken der Temperatur

In Abbildung 5.16 ist zu sehen, wie oft die Zeitbedingung überschritten wurde und wie groß die Überschreitung im Schnitt war.
Es ist zu erkennen, dass bei $N_{It} = 1000$ die Zeitbedingung nur sehr selten erfüllt wurde. Mit der Starttemperatur 15 wurde sie am häufigsten eingehalten, aber auch dabei lediglich in 17 Versuchen. Die kleinsten Überschreitungen von $t_{max,C}$ wurden mit $T_0 = 25$ ermittelt. Bei 2000 Iterationen pro Temperatur ist ein ähnlicher Verlauf der Werte zu sehen. Bei $T_0 = 20$ wurden die meisten Versuche mit Einhaltung der Zeitbedingung berechnet, allerdings sind die Überschreitungen bei $T_0 = 30$ am kleinsten. Diese Ergebnisse lassen sich mit der Instabilität des Verfahrens begründen.

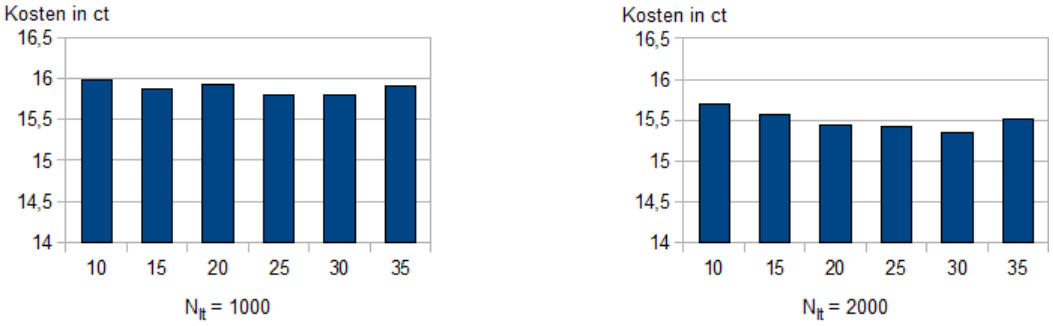

Abb. 5.17: Vergleich der durchschnittlichen Kosten pro Chip mit Simulated Annealing und logarithmischem Temperaturverlauf

In Abbildung 5.17 sind zwei Diagramme dargestellt. Darin sind auf der Ordinate die Kosten pro Chip in Cent aufgetragen, auf der Abszisse die Startwerte. Die Kosten bei 1000 Iterationen pro Temperatur unterscheiden sich nur leicht. Die besten Werte wurden mit $N_{It} = 1000$ bei $T_0 = 25$ erzielt. Bei $N_{It} = 2000$ sind die Kosten insgesamt niedriger. Für den Startwert 30 wurden für diese Anzahl an Iterationen die besten

Ergebnisse berechnet.
Abschließend ist zu bemerken, dass das logarithmische Absenken der Temperatur recht instabil ist. Die Zielfunktionswerte schwanken stark und die Zeitbedingung wird gerade bei $N_{It} = 1000$ oft verletzt. Die besten Lösungen wurden mit 2000 Iterationen pro Temperatur und $T_0 = 20$ berechnet, auch wenn bei dieser Konfiguration die Kosten etwas größer waren.

Vergleich aller drei Abkühlungskurven

Im Folgenden sollen die drei Methoden zum Absenken der Temperatur miteinander verglichen werden.
Zunächst ist anzumerken, dass beim exponentiellen Absenken der Temperatur mit einer großen Bandbreite von Startwerten vergleichbare Ergebnisse erzielt wurden. Beim logarithmischen Absenken war eine große Schwankung der Ergebnisse und damit eine deutliche Instabilität des Verfahrens zu sehen. Gerade bei 1000 Iterationen pro Temperatur wurde dieses Verhalten deutlich.
Nun soll untersucht werden, mit welcher Methodik die besten Ergebnisse erzielt wurden. Dazu wird von jedem Verfahren zur Absenkung der Temperatur der Startwert ausgewählt, mit dem bei $N_{It} = 1000$ bzw. $N_{It} = 2000$ der beste durchschnittliche Zielfunktionswert ermittelt wurde. Diese Werte sind in der folgenden Abbildung 5.18 dargestellt.

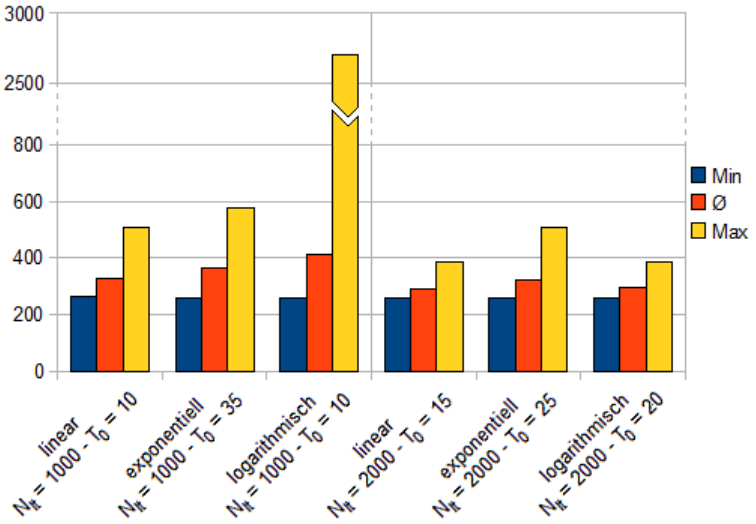

Abb. 5.18: Vergleich der Zielfunktionswerte für verschiedene Abkühlungskurven mit Simulated Annealing

Wie dem Diagramm 5.18 zu entnehmen ist, wurden bei $N_{It} = 1000$ mit linearem Absenken der Temperatur im Schnitt die besten Ergebnisse ermittelt. Bei 2000 Iterationen pro Temperatur wurden die besten Zielfunktionswerte mit linearem und logarithmischem Kurvenverlauf berechnet. Diese scheinen - bei alleiniger Betrachtung dieses Diagramms - auch ähnlich stabil zu sein. Allerdings wurden beim linearen Absenken auch mit anderen Temperaturen bei $N_{It} = 2000$ vergleichbar gute Ergebnisse ermittelt. Beim logarithmischen Temperaturverlauf hingegen wurde nicht einmal die Zeitbedingung in der Abschätzung ohne TSVs immer eingehalten. Daraus lässt sich schließen, dass von den drei hier verglichenen Methoden das lineare Absenken bei 1000 und 2000 Iterationen pro Temperatur die besten durchschnittlichen Zielfunktionswerte ermittelt und dass dieses Verfahren am stabilsten ist.

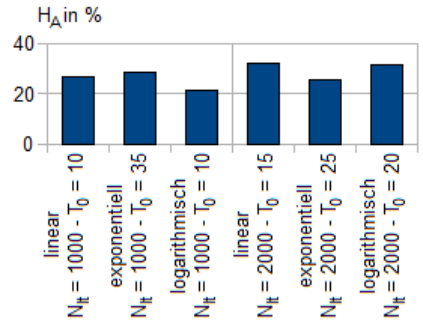

Abb. 5.19: durchschnittliche Annahmehäufigkeit für verschiedene Abkühlungskurven mit Simulated Annealing

In Abbildung 5.19 ist die durchschnittliche Häufigkeit, mit der Nachbarschaftslösungen angenommen wurden, dargestellt. Es ist zu sehen, dass bei allen Verfahren ungefähr 30% der Lösungen angenommen wurden. Gerade die Methoden des lineares Absenken mit $T_0 = 15$ und des logarithmischer Temperaturverlauf mit $T_0 = 20$ nehmen ungefähr gleich viele Lösungen an (32%). Dieser Wert scheint ein guter Prozentsatz für die Annahmehäufigkeit zu sein.

In Abbildung 5.20 ist dargestellt, wie oft die Zeitbedingung erfüllt wurde und wie groß die durchschnittliche Überschreitung der maximal zulässigen Zeit war. Es ist zu sehen, dass bei 1000 Iterationen pro Temperatur mit dem linearen Absenken die meisten Lösungen mit zulässiger Zeitbedingung errechnet wurden und auch die durchschnittliche Überschreitung der maximal zulässigen Zeit am kleinsten war. Bei $N_{It} = 2000$ wurden mit dem logarithmischen Absenken der Temperatur die meisten zulässigen Lösungen berechnet, allerdings schneidet das lineare Absenken

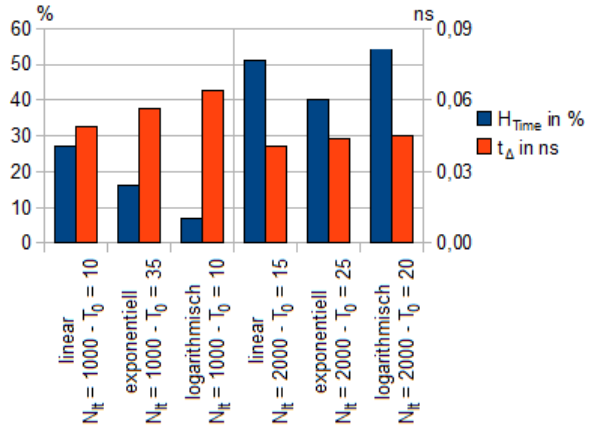

Abb. 5.20: Einhaltung der Zeitbedingung und durchschnittliche Überschreitung von $t_{max,C}$ für verschiedene Abkühlungskurven mit Simulated Annealing

kaum schlechter ab.

Da das lineare Absenken deutlich stabilere Ergebnisse geliefert hat, ist es die beste der drei hier vorgestellten Methoden zum Absenken der Temperatur. Dabei sind die Lösungen bei 2000 Iterationen pro Temperatur für alle Startwerte gut, die besten Werte wurden aber mit $T_0 = 15$ ermittelt. Da bei $N_{It} = 1000$ die besten Zielfunktionswerte mit $T_0 = 10$ berechnet wurden, ist anzunehmen, dass mit steigender Anzahl an Iterationen der Startwert leicht erhöht werden sollte, um optimale Ergebnisse zu erhalten.

5.4 Threshold Accepting

5.4.1 Der Algorithmus

Dieses Verfahren wurde 1989 von Gunter Dueck und Tobias Scheuer [DS90] entwickelt. Bei dieser Methode wird die Nachbarschaftslösung $l_N \in F_N(l_i)$ angenommen, wenn sie nicht viel schlechter als die aktuelle Lösung ist:

Sei $S_i \geq 0$, $i = 0, 1, 2 \ldots$ eine monoton fallende Folge von Schwellenwerten, für die gilt: $\forall i \geq 0 : S_i \geq S_{i+1}$. Dann kann das Annahmekriterium der Nachbarschaftslösung wie folgt beschrieben werden:

$$l_{i+1} \leftarrow l_N \in F_N(l_i) \text{ genau dann, wenn } f(l_N) - f(l_i) \leq S_i.$$

Die Nachbarschaftslösung wird also genau dann angenommen, wenn deren Funktionswert höchstens um den Wert S_i größer als die Güte der aktuellen Lösung ist.

Auch hier sollte die Schwelle am Anfang recht hoch sein und mit der Zeit langsam absinken. Um am Ende ein lokales Minimum zu erreichen, sollten die letzten Schwellenwerte bei Null liegen. Es ist darauf zu achten, die Schwellen weder zu groß, noch zu klein zu wählen, um möglichst gute Lösungen zu finden.
Mit diesem Algorithmus kann schneller aus lokalen Minima herausgefunden werden als beim Simulated Annealing, da „kleine“ Verschlechterungen immer angenommen werden. Allerdings können beim Simulated Annealing, wenn auch nur mit kleiner Wahrscheinlichkeit, auch größere Verschlechterungen angenommen und somit auch am Ende des Algorithmus' lokale Minima überwunden werden.

5.4.2 Ergebnisse

Bei diesem Algorithmus wurde ähnlich wie bei Simulated Annealing vorgegangen. Es wurde eine Liste von Schwellenwerten $[S_i, i = 1, 2, \ldots 50]$ erzeugt und mit jedem Schwellenwert N_{It} Iterationen ausgeführt. Dabei wurden wiederum drei verschiedene Kurvenverläufe mit verschiedenen Startwerten S_0 untersucht. Die Schwellenwerte $S_i,\ i = 1, 2, \ldots 45$ der verschiedenen Varianten wurden wie folgt berechnet:

- linear: $S_i = S_0 \cdot (1 - \frac{i-1}{45})$
- exponentiell: $S_i = S_0 \cdot 0.9^{i-1}$
- kubisch: $S_i = S_0 \cdot (1 - \frac{(i-1)^3}{45^3})$

Die letzten fünf Schwellenwerte wurden auch bei Threshold Accepting auf Null gesetzt, um am Ende möglichst in ein lokales Minimum zu laufen.

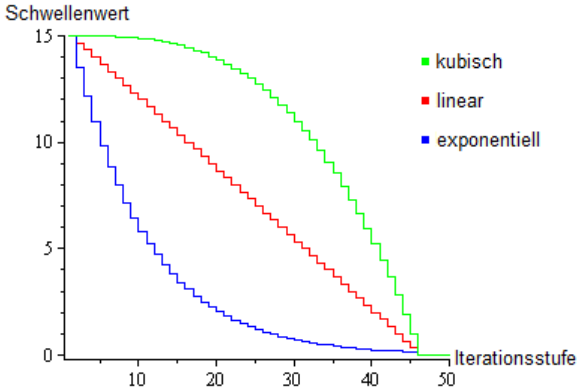

Abb. 5.21: Drei Varianten für die Schwellenwerte

In Abbildung 5.21 wurden die drei verschiedenen Kurvenverläufe für die Schwellenwerte mit $S_0 = 15$ dargestellt.
Bei jeder der drei Variationen wurde mit verschiedenen Startwerten S_0 begonnen. Dabei wurde versucht, auszuloten, in welchem Bereich gute Lösungen ermittelt werden und ab wann die Schwellenwerte zu klein bzw. zu groß sind.

Lineares Absenken der Schwellenwerte

Im Folgenden wird dargestellt, welche Ergebnisse mit dem linearen Absenken der Schwellenwerte erzielt wurden. Dabei wurden für verschiedene Startwerte S_0 mit 1000 und 2000 Iterationen pro Schwellenwert jeweils 100 Berechnungen durchgeführt. Die Ergebnisse sind in der folgenden Tabelle dargestellt. Notiert sind die durchschnittliche Annahmehäufigkeit H_A, die Zielfunktionswerte, die Kosten, die TSV-Anzahl N_{TSV}, die Häufigkeit H_{Time}, mit der die berechnete Lösung nach Einfügen der TSVs die Zeitbedingung einhält und die durchschnittliche Überschreitung t_Δ der maximal zulässigen Zeit, die ein Signal innerhalb des Chips benötigen darf.

S_0	H_A	Zielfunktionswert			Kosten	$\varnothing N_{TSV}$	H_{Time}	t_Δ
	∅ in %	∅	Min	Max	∅ in ct		in %	in ns
$N_{It} = 1000$								
5	20	451	264	594	16.25	1891	2	0.082
10	27	364	263	553	15.83	1449	20	0.056
15	32	328	264	432	15.63	1263	25	0.045
20	37	364	264	2686	15.58	1210	35	0.047
25	42	442	263	2651	15.67	1266	26	0.050
30	46	595	264	2795	15.78	1356	21	0.060
$N_{It} = 2000$								
5	20	356	262	598	16.08	1789	7	0.072
10	27	350	262	557	15.67	1381	25	0.048
15	32	297	262	380	15.38	1090	63	0.039
20	37	287	262	392	15.33	1040	71	0.033
25	41	298	261	413	15.37	1102	60	0.037
30	45	310	262	411	15.45	1175	50	0.043

Tab. 5.4: Ergebnisse mit Threshold Accepting und linear sinkenden Schwellenwerte aus 100 Versuchen

In Tabelle 5.4 ist zu sehen, dass die Annahmehäufigkeit H_A weitestgehend unabhängig von der Anzahl an Iterationen ist.
Wie der Tabelle zu entnehmen ist, wurden die kleinsten Zielfunktionswerte bei $N_{It} = 1000$ mit $S_0 = 15$ berechnet. Die kleinsten Kosten und die meisten zulässigen Lösungen bei 1000 Iterationen pro Schwellenwert wurden allerdings mit $S_0 = 20$

ermittelt. Das liegt vermutlich daran, dass in diesem Fall einige Lösungen berechnet wurden, bei denen die Zeitbedingung in der Abschätzung ohne TSVs nicht eingehalten wurde. Mit dem Startwert 20 werden also im Schnitt bessere Ergebnisse ermittelt, allerdings schwanken die Lösungen stärker, sodass der durchschnittliche Zielfunktionswert schlechter ausfällt. Bei 2000 Iterationen pro Schwellenwert wurden die besten Ergebnisse mit $S_0 = 20$ berechnet.

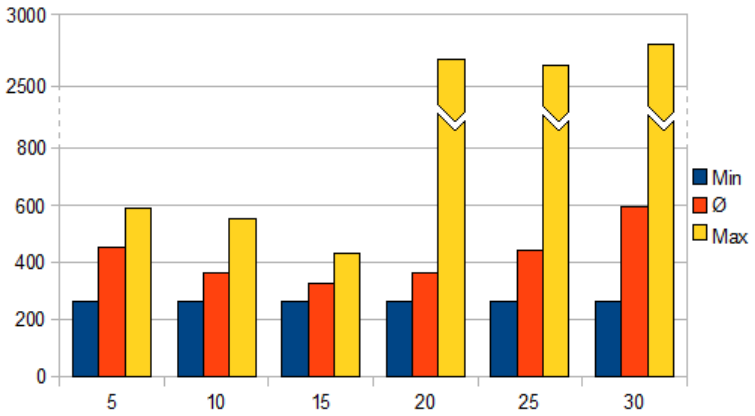

Abb. 5.22: Vergleich der berechneten Zielfunktionswerte mit Threshold Accepting, $N_{It} = 1000$ und linearem Absenken der Schwellenwerte

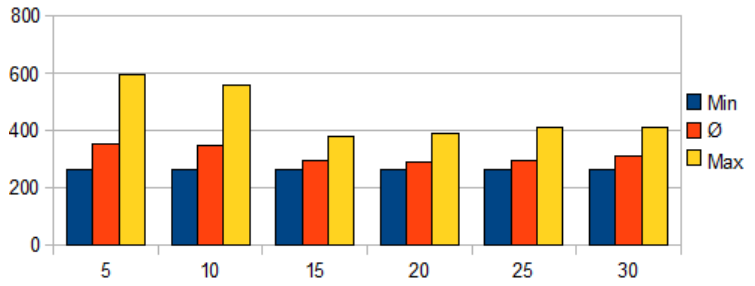

Abb. 5.23: Vergleich der berechneten Zielfunktionswerte mit Threshold Accepting, $N_{It} = 2000$ und linearem Absenken der Schwellenwerte

In den Abbildungen 5.22 und 5.23 werden die berechneten Zielfunktionswerte graphisch dargestellt. Auf der Abszisse sind die Startwerte abgetragen, auf der Ordinate der minimale, durchschnittliche und der maximale Zielfunktionswert. Deutlich ist zu sehen, dass bei 1000 Iterationen pro Schwellenwert die Zielfunktionswerte stärker schwanken als bei $N_{It} = 2000$. Das Verfahren gewinnt also mit zunehmender Anzahl an Iterationen an Stabilität. Bei $N_{It} = 1000$ ist eine deutliche Tendenz zu sehen, dass mit dem Startwert 15 im Schnitt die besten Zielfunktionswerte erreicht werden. Der durchschnittliche und maximale Zielfunktionswert steigt mit zunehmenden und

abnehmenden Startwerten. Der kleinste erhaltene Zielfunktionswert ist bei allen Startwerten ungefähr gleich.
Bei 2000 Iterationen pro Temperatur wurden mit den Startwerten 15 bis 25 sehr ähnliche Ergebnisse ermittelt. Mit zunehmender Anzahl an Iterationen steigt also nicht nur die Stabilität des Verfahrens, es wird auch zunehmend robuster gegenüber der Wahl des Startwerts.

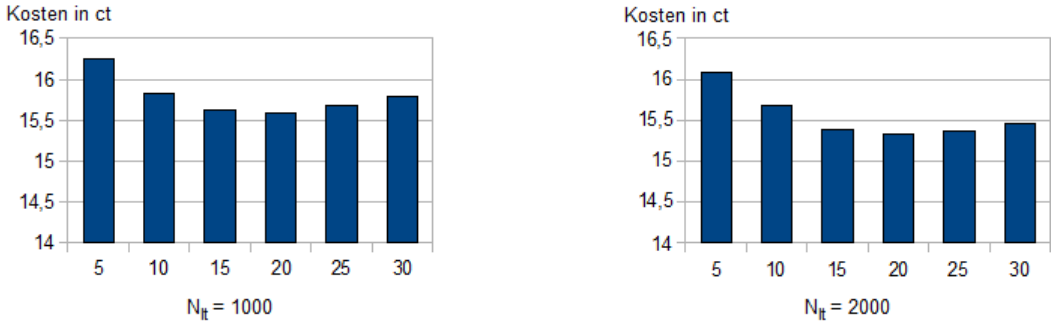

Abb. 5.24: Vergleich der durchschnittlichen Kosten pro Chip mit Threshold Accepting und linearem Ansenken der Schwellenwerte

In Abbildung 5.24 sind zwei Diagramme abgebildet, die Aufschluss über die durchschnittlichen Kosten pro Chip in Abhängigkeit vom gewählten Startwert geben sollen. Dabei sind auf der Abszisse die Kosten und auf der Ordinate die Startwerte abgetragen.
In dem linken Diagramm sind die Kosten zu sehen, die bei 1000 Iterationen pro Schwellenwert im Schnitt ermittelt wurden. Es ist deutlich zu erkennen, dass die kleinsten Kosten bei $S_0 = 20$ anfallen. In dem rechten Diagramm ist ein ähnlicher Verlauf der Kosten in Abhängigkeit von den Startwerten zu erkennen. Die kleinsten Werte wurden wiederum mit $S_0 = 20$ ermittelt. Die Kosten liegen aber bei 2000 Iterationen deutlich unter denen mit $N_{It} = 1000$.
Abbildung 5.25 zeigt zwei Diagramme, in denen die Häufigkeit H_{Time}, mit der die Lösung nach dem Einfügen der TSVs zulässig war und die durchschnittliche Überschreitung von $t_{max,C}$ abgebildet sind. Auf der Abszisse sind die Startwerte abgetragen. H_{Time} befindet sich jeweils auf der linken Ordinate. Die durchschnittliche Überschreitung von $t_{max,C}$ der unzulässigen Lösungen ist auf der rechten Ordinate abgetragen. Beim Vergleich der Diagramme fällt auf, dass bei 2000 Iterationen deutlich mehr zulässige Lösungen als bei $N_{It} = 1000$ ermittelt wurden. Auch die durchschnittliche Überschreitung der maximal zulässigen Zeit ist deutlich niedriger.

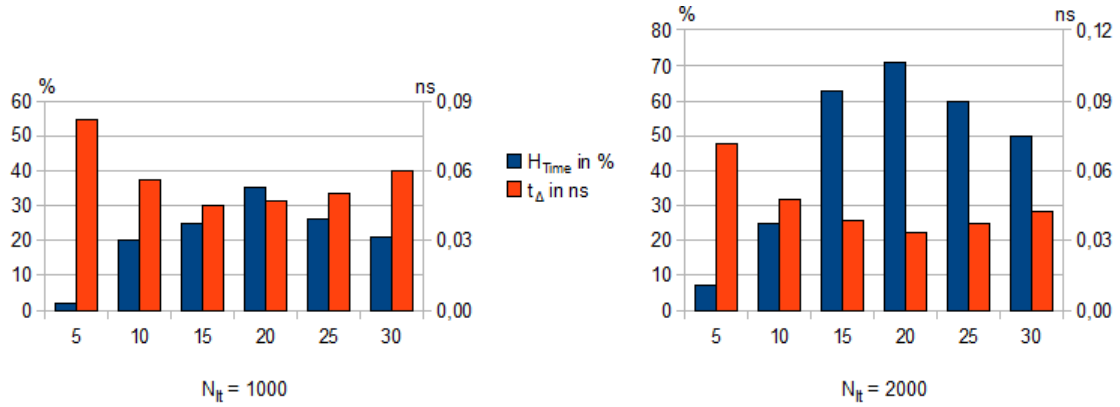

Abb. 5.25: Einhaltung der Zeitbedingung und durchschnittliche Überschreitung von $t_{max,C}$ mit Threshold Accepting und linearem Absenken der Schwellenwerte

Allerdings wird deutlich, dass bei 1000 und 2000 Iterationen pro Schwellenwert eine fast identische Entwicklung der Werte zu sehen ist. Bis zum Startwert 20 steigt H_{Time} an, erreicht dort seinen Höhepunkt und sinkt dann wieder langsam ab. Umgekehrt verhält sich die durchschnittliche Überschreitung von $t_{max,C}$. Diese sinkt bis $S_0 = 20$ und nimmt danach wieder zu.

Abschließend kann gesagt werden, dass mit $S_0 = 20$ die besten Ergebnisse erzielt werden. Auch ist zu beobachten, dass mit den Startwerten 15 und 25 bei $N_{It} = 2000$ kaum schlechtere Lösungen ermittelt wurden. Eine weitere Steigerung der Anzahl an Iterationen sollte das Verfahren noch robuster gegenüber der Wahl des Startwerts werden lassen.

Exponentielles Absenken der Schwellenwerte

Im Folgenden werden die Ergebnisse für das exponentielle Absenken der Schwellenwerte vorgestellt. Die Resultate werden zunächst tabellarisch und anschließend graphisch dargestellt. Dabei wird die durchschnittliche Annahmehäufigkeit H_A, die ermittelten Zielfunktionswerte, die Kosten pro Chip, die TSV-Anzahl N_{TSV}, die Häufigkeit H_{Time}, mit der die Endlösung nach Einfügen der TSVs die Zeitbedingung einhält und die durchschnittliche Überschreitung von $t_{max,C}$ bei Verletzung der Zeitbedingung untersucht.

Der Tabelle 5.5 ist zu entnehmen, dass die durchschnittliche Annahmehäufigkeit unabhängig von der Anzahl an Iteration ist. Auch ist zu sehen, dass die minimalen

S_0	H_A	Zielfunktionswert			Kosten	$\varnothing N_{TSV}$	H_{Time}	t_Δ
	$\varnothing$ in %	$\varnothing$	Min	Max	$\varnothing$ in ct		in %	in ns
$N_{It} = 1000$								
5	14	513	262	2808	16.38	2084	1	0.094
15	21	369	262	563	15.76	1489	11	0.055
25	26	365	263	570	15.79	1468	9	0.053
35	30	355	263	563	15.76	1397	27	0.061
45	33	386	263	2663	15.80	1446	25	0.061
65	38	393	263	2635	15.86	1484	14	0.058
85	42	389	264	2812	15.72	1347	26	0.051
105	45	458	264	2838	15.88	1469	14	0.058
125	48	521	264	2833	15.88	1448	17	0.059
$N_{It} = 2000$								
5	14	445	261	630	16.09	1862	8	0.077
15	21	343	261	539	15.57	1353	24	0.044
25	26	328	261	524	15.50	1257	38	0.046
35	28	325	262	506	15.51	1245	40	0.046
45	33	322	262	511	15.48	1229	43	0.046
65	38	320	262	538	15.51	1236	38	0.043
85	42	323	262	512	15.51	1245	44	0.044
105	45	329	262	503	15.56	1279	33	0.044
125	48	364	262	2686	15.60	1351	27	0.047

Tab. 5.5: Ergebnisse mit Threshold Accepting und exponentiell sinkenden Schwellenwerte aus 100 Versuchen

Zielfunktionswerte recht konstant sind. Sie liegen bei allen Versuchen zwischen 261 und 264. Die größten, berechneten Zielfunktionswerte schwanken dagegen stark. Bei den Versuchen mit den Startwerten 15 bis 35 und $N_{It} = 1000$ wurde die Zeitbedingung in der Abschätzung ohne TSVs immer eingehalten. Allerdings werden bei größeren Startwerten mehr zulässige Lösungen generiert. Das Verfahren ist also bei 1000 Iterationen pro Schwellenwert und $S_0 \geq 45$ recht instabil. Die Verdopplung der Anzahl an Iterationen bewirkte eine Verbesserung der durchschnittlichen Zielfunktionswerte und eine Stabilisierung des Verfahrens. Auch die Anzahl an zulässigen Lösungen stieg an.

In den Abbildungen 5.26 und 5.27 ist jeweils ein Diagramm für die ermittelten minimalen, durchschnittlichen und maximalen Zielfunktionswerte dargestellt. Auf der Abszisse sind die Startwerte S_0 abgetragen, auf der Ordinate die berechneten Zielfunktionswerte.

Dem Diagramm 5.26 ist zu entnehmen, dass die Zielfunktionswerte bei 1000 Iterationen pro Schwellenwert und den Startwerten $S_0 \geq 20$ stark schwanken. Die besten Lösungen wurden mit dem Startwert $S_0 = 35$ ermittelt. Die Ergebnisse der Startwerte 15 und 25 sind aber kaum schlechter. Mit steigendem Startwert verliert

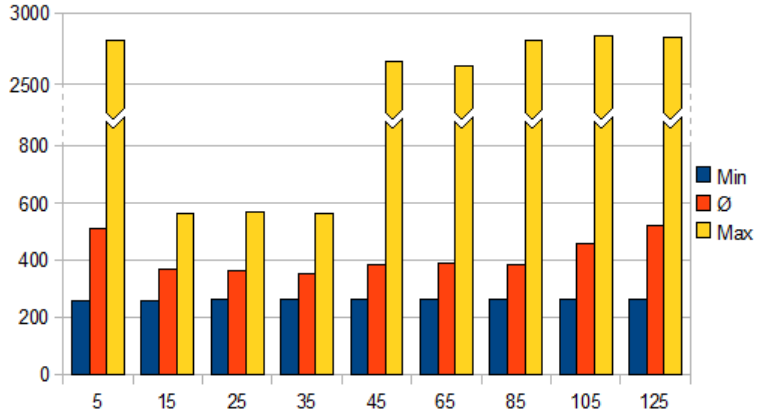

Abb. 5.26: Vergleich der berechneten Zielfunktionswerte mit Threshold Accepting, $N_{It} = 1000$ und exponentiell sinkenden Schwellenwerten

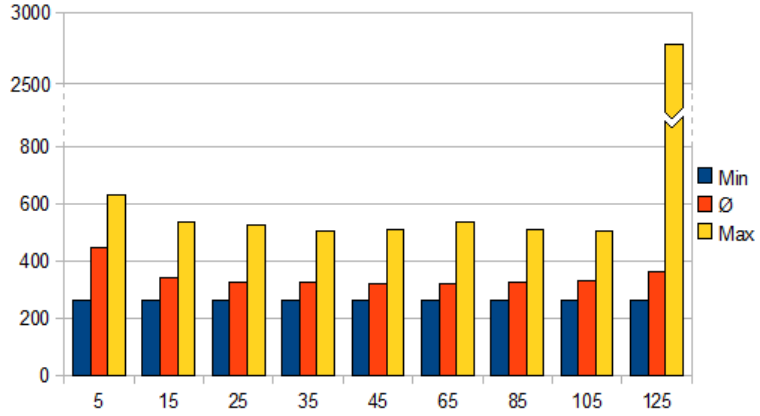

Abb. 5.27: Vergleich der berechneten Zielfunktionswerte mit Threshold Accepting, $N_{It} = 2000$ und exponentiell sinkenden Schwellenwerten

das Verfahren zunehmend an Stabilität, was erhöhte durchschnittliche Zielfunktionswerte zur Folge hat. Das liegt vermutlich daran, dass der Algorithmus bei den ersten großen Temperaturen zu viele Lösungen annimmt und erst später optimiert. Damit geht wichtige Iterationszeit verloren. Auch zu kleine Startwerte führen zu schlechten Ergebnissen, wie aus den Zielfunktionswerten mit $S_0 = 5$ geschlossen werden kann. Die erhaltenen Werte sind im Schnitt deutlich schlechter. Hier hatte der Algorithmus zu wenig Spielraum, um lokale Minima zu überwinden.

In Abbildung 5.27 fällt auf, dass die Zielfunktionswerte auf fast dem gesamten Spektrum an Startwerten sehr ähnlich sind. Lediglich bei $S_0 = 5$ sind die Lösungen im Schnitt schlechter. Bei $S_0 = 125$ wurde ein sehr großer, schlechtester Zielfunktionswert berechnet, auch der durchschnittliche Zielfunktionswert ist leicht gestiegen.

Die Ergebnisse der Startwerte 15 bis 105 bei 2000 Iterationen pro Schwellenwert

sind recht konstant. Das exponentielle Absenken der Schwellenwerte führt also dazu, dass mit einer großen Bandbreite von Startwerten vergleichbare Ergebnisse erzeugt werden.

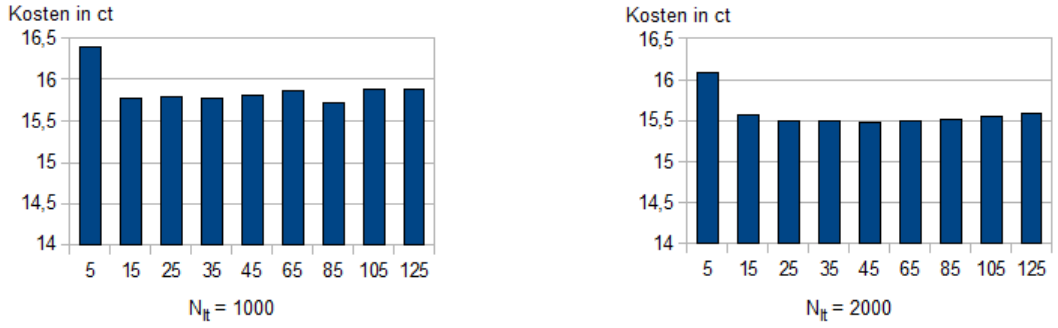

Abb. 5.28: Vergleich der durchschnittlichen Kosten pro Chip mit Threshold Accepting und exponentiellem Absenken der Schwellenwerte

In Abbildung 5.28 wird die Entwicklung der Kosten mit steigendem Startwert für 1000 und 2000 Iterationen pro Schwellenwert in zwei Diagrammen dargestellt. Auf der Abszisse sind jeweils die Startwerte abgetragen, auf der Ordinate die Kosten in Cent.

In beiden Diagrammen ist zu sehen, dass die größten Kosten bei $S_0 = 5$ auftreten. Mit diesem Startwert werden die schlechtesten Lösungen berechnet. Im linken Diagramm ist zu sehen, dass bei 1000 Iterationen pro Schwellenwert die Kosten mit zunehmenden Startwert wieder leicht ansteigen. Eine Ausnahme ist dabei $S_0 = 85$, dort wurden mit 15.72 ct die kleinsten Kosten ermittelt. In dem rechten Diagramm ist bei $N_{It} = 2000$ ein leichtes Absenken der durchschnittlichen Kosten pro Chip bis $S_0 = 45$ und danach ein leichtes Ansteigen der Kosten zu sehen. Die Unterschiede sind aber sehr klein. Bei $S_0 = 45$ werden durchschnittliche Kosten von 15.48 ct ermittelt. Das sind die kleinsten Werte aus allen Startwerten bei 2000 Iterationen pro Schwellenwert.

In Abbildung 5.29 sind die durchschnittliche Überschreitung der maximal zulässigen Zeit, die ein Signal innerhalb des Chips verweilen darf und die Häufigkeit, mit der die Zeitbedingung auch nach dem Einfügen der TSVs eingehalten wird, für 1000 und 2000 Iterationen pro Schwellenwert in je einem Diagramm dargestellt.

Dem linken Diagramm ist zu entnehmen, dass die Zeitbedingung häufig verletzt wurde. Bei keinem Startwert waren mehr als 30% der Lösungen nach dem Einfügen der TSVs zulässig. Auch die durchschnittliche Überschreitung der maximal zulässigen Zeit ist mit 0.06 ns recht hoch. Insgesamt kann festgehalten werden, dass die meisten

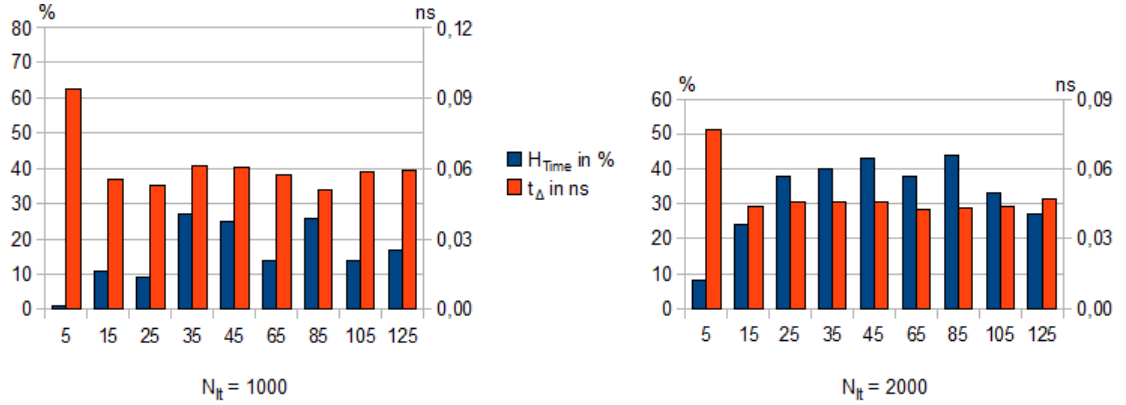

Abb. 5.29: Einhaltung der Zeitbedingung und durchschnittliche Überschreitung von $t_{max,C}$ mit Threshold Accepting und exponentiell sinkenden Schwellenwerten

zulässigen Lösungen mit $S_0 = 35$ bis $S_0 = 85$ erzeugt wurden. Die Überschreitung t_Δ der maximal zulässigen Zeit ist bei diesen Startwerten aber nicht kleiner.
Bei 2000 Iterationen pro Schwellenwert werden deutlich mehr zulässige Lösungen berechnet. Das ist gut in dem rechten Diagramm zu erkennen. Mit zu- und abnehmende Startwert sinkt die Anzahl an Lösungen, bei denen auch nach dem Einfügen der TSVs die Zeitbedingung eingehalten wird, deutlich ab. Die durchschnittliche Überschreitung von $t_{max,C}$ ist dagegen - außer bei $S_0 = 5$ - bei allen Startwerten mit etwa 0.045 ns ungefähr gleich groß.
Abschließend kann gesagt werden, dass die Bandbreite an Startwerten, mit denen eine gute Lösung berechnet wird, sehr groß ist. Deswegen ist diese Abkühlungskurve gut geeignet, um auszuloten, in welchem Bereich die besten Startwerte liegen.

Kubisches Absenken der Schwellenwerte

Diese Methode zum Absenken der Schwellenwerte wurde ausgewählt, um zu untersuchen, wie sich der Algorithmus verhält, wenn am Anfang die Schwellenwerte recht konstant sind und kaum absinken und am Ende schnell auf Null gesenkt werden. Um möglichst in einem lokalen Minimum zu enden, wurden die letzten fünf Temperaturen wieder auf Null gesetzt.
Im Folgenden werden die Ergebnisse tabellarisch dargestellt. Darin sind die durchschnittliche Häufigkeit H_A, mit der Lösungen angenommen wurden, die berechneten Zielfunktionswerte, die Kosten, die TSV-Anzahl N_{TSV}, die Häufigkeit H_{Time}, mit der

die Lösung nach dem Einfügen der TSVs zulässig ist und mit t_Δ die Überschreitung der maximal zulässigen Zeit $t_{max,C}$ zu sehen.

T_0	H_A	Zielfunktionswert			Kosten	$\varnothing N_{TSV}$	H_{Time}	t_Δ
	$\varnothing$ in %	$\varnothing$	Min	Max	$\varnothing$ in ct		in %	in ns
$N_{It} = 1000$								
5	24	546	265	2824	16.21	1798	5	0.078
7.5	29	482	264	2816	15.90	1472	12	0.060
10	33	453	263	2740	15.74	1321	24	0.054
12.5	36	370	264	2631	15.67	1242	30	0.046
15	39	397	264	2639	15.60	1141	29	0.038
17.5	42	487	264	2802	15.57	1135	31	0.040
20	46	802	264	2742	15.77	1251	17	0.052
$N_{It} = 2000$								
5	24	397	262	584	15.93	1612	16	0.063
7.5	29	335	262	546	15.62	1305	36	0.048
10	33	307	262	517	15.49	1151	51	0.040
12.5	36	297	262	383	15.42	1090	53	0.034
15	38	279	262	379	15.35	986	70	0.026
17.5	41	278	262	380	15.28	983	73	0.027
20	46	297	262	384	15.39	1092	55	0.039

Tab. 5.6: Ergebnisse mit Threshold Accepting und kubischen Absenken der Schwellenwerte aus 100 Versuchen

Der Tabelle 5.6 ist zu entnehmen, dass die durchschnittliche Häufigkeit, mit der Lösungen angenommen wurde, wiederum unabhängig von der Anzahl an Iterationen ist.

Die minimalen, berechneten Zielfunktionswerte liegen wiederum alle im Bereich 262 bis 265. Theoretisch kann also mit allen Verfahren eine ähnlich gute Lösung ermittelt werden. Im Schnitt wurde bei $N_{It} = 1000$ mit $S_0 = 12.5$ die besten Lösungen ermittelt. Bei 2000 Iterationen pro Schwellenwert wurden die besten durchschnittlichen Ergebnisse mit $S_0 = 17.5$ berechnet.

Mit $N_{It} = 1000$ wird in der Abschätzung ohne TSVs die Zeitbedingung bei jedem Startwert wenigstens einmal verletzt. Deswegen liegt der größte, berechnete Zielfunktionswert auch deutlich über 2500. Die Verdopplung der Anzahl an Iterationen bewirkt aber eine Stabilisierung des Verfahrens, sodass die Zeitbedingung in der Abschätzung immer eingehalten wird und auch nach dem Einfügen der TSVs ein großer Teil der Lösungen zulässig ist.

In den Diagrammen aus den Abbildungen 5.30 und 5.31 sind die minimalen, durchschnittlichen und maximalen Zielfunktionswerte aus 100 Versuchen mit den verschiedenen Startwerten dargestellt. Auf der Ordinate sind dabei die Startwerte S_0

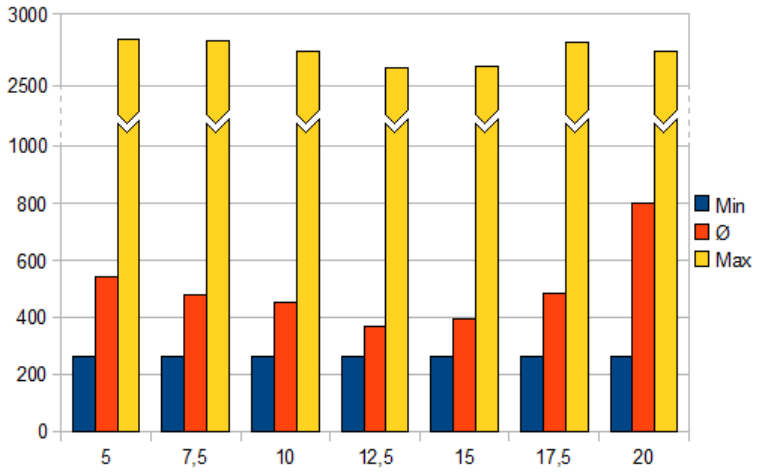

Abb. 5.30: Vergleich der berechneten Zielfunktionswerte mit Threshold Accepting, $N_{It} = 1000$ und kubischen Absenken der Schwellenwerte

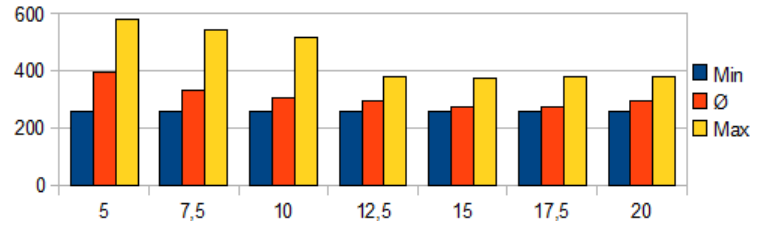

Abb. 5.31: Vergleich der berechneten Zielfunktionswerte mit Threshold Accepting, $N_{It} = 2000$ und kubischen Absenken der Schwellenwerte

abgetragen, auf der Abszisse die berechneten Zielfunktionswerte.

Beim Vergleich der beiden Diagramm fällt auf, dass bei 2000 Iterationen pro Schwellenwert deutlich bessere Ergebnisse erzielt werden als bei $N_{It} = 1000$.

Im Diagramm 5.30 ist gut zu erkennen, wie die durchschnittlichen Zielfunktionswerte mit steigendem Startwert bis $S_0 = 12.5$ kontinuierlich abnehmen und danach wieder stark ansteigen. Bei 2000 Iterationen pro Schwellenwert ein ähnlicher Verlauf der Werte zu sehen. Allerdings sind die Unterschiede zwischen den durchschnittlichen Zielfunktionswerten der verschiedenen Startwerte deutlich kleiner. Die besten Ergebnisse werden mit $S_0 = 15$ und $S_0 = 17.5$ erzielt. Die durchschnittlichen Zielfunktionswerte dieser Startwerte betragen 279 und 278.

Das Verfahren ist bei längerer Laufzeit also stabiler und robuster gegenüber der Wahl des Startwerts.

In den beiden Diagrammen aus Abbildung 5.32 ist dargestellt, wie oft die Zeitbedingung nach dem Einfügen der TSVs eingehalten wurde und wie groß die durchschnittliche Überschreitung von $t_{max,C}$ war. Es ist gut zu sehen, dass bei $N_{It} = 1000$

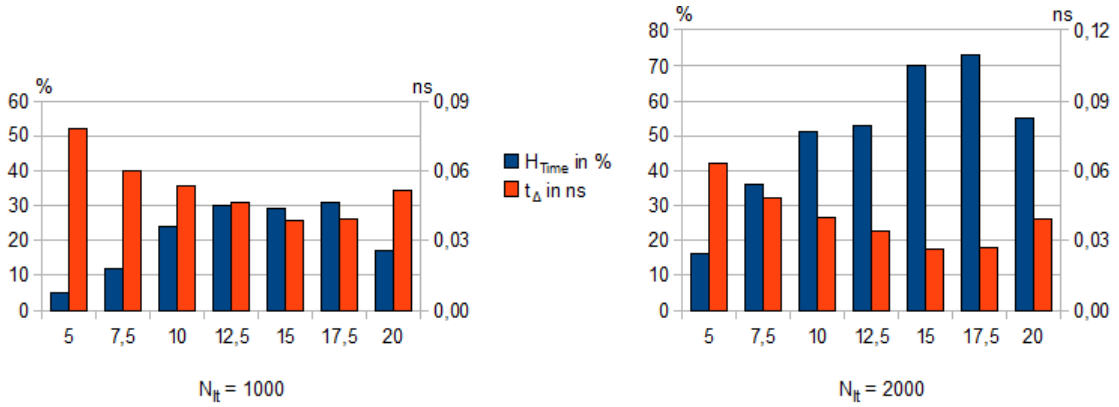

Abb. 5.32: Einhaltung der Zeitbedingung und durchschnittliche Überschreitung von $t_{max,C}$ mit Threshold Accepting und kubischen Absenken der Schwellenwerte

die Zeitbedingung mit den Startwerten 12.5, 15 und 17.5 am häufigsten eingehalten wurde, jeweils rund 30 mal. Bei 2000 Iterationen pro Schwellenwert wurden die meisten zulässigen Lösungen (72%) mit $S_0 = 17.5$ erzielt. Dabei waren mit den Startwerten 10 bis 20 jeweils mehr als 50% der Lösungen zulässig.
Auch die durchschnittliche Überschreitung von $t_{max,C}$ ist bei 2000 Iterationen pro Schwellenwert deutlich kleiner als bei $N_{It} = 1000$.

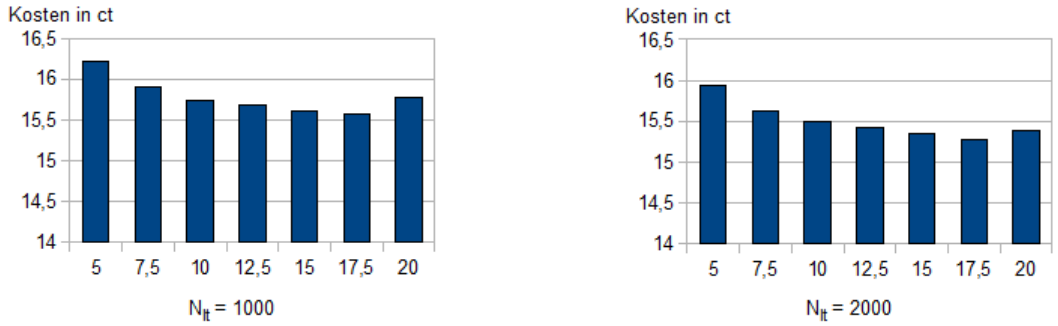

Abb. 5.33: Vergleich der durchschnittlichen Kosten pro Chip mit Threshold Accepting und kubischen Absenken der Schwellenwerte

In Abbildung 5.33 sind zwei Diagramme dargestellt, in denen die durchschnittlichen Kosten pro Chip in Abhängigkeit von den Startwerten aufgetragen sind. Auf den ersten Blick ist zu sehen, dass die Kosten bei 2000 Iterationen pro Schwellenwert deutlich kleiner sind als bei $N_{It} = 1000$. Auch ist in beiden Diagrammen gut zu sehen, dass die Kosten bis $S_0 = 17.5$ kontinuierlich sinken und dort mit 15.57 ct und 15.28 ct ihr Minimum erreichen. Danach steigen die Kosten wieder an.
Abschließend kann gesagt werden, dass Threshold Accepting mit kubisch sinken-

den Schwellenwerten bei 1000 Iterationen pro Schwellenwert recht instabil ist. Bei $N_{It} = 2000$ werden aber relativ konstante Lösungen berechnet. Die besten Lösungen werden mit $S_0 = 17.5$ erzielt. Mit diesem Startwert sind 72% der Lösungen auch nach dem Einfügen der TSVs noch zulässig.

Vergleich aller drei Methoden zum Senken der Schwellenwerte

Im Folgenden werden die drei hier vorgestellten Methoden zum Absenken der Schwellenwerte miteinander verglichen. Dabei wurden immer die Startwerte mit dem niedrigsten durchschnittlichen Zielfunktionswert herausgegriffen. Es soll untersucht werden, wie die drei Abkühlungskurven im direkten Vergleich zueinander abschneiden.

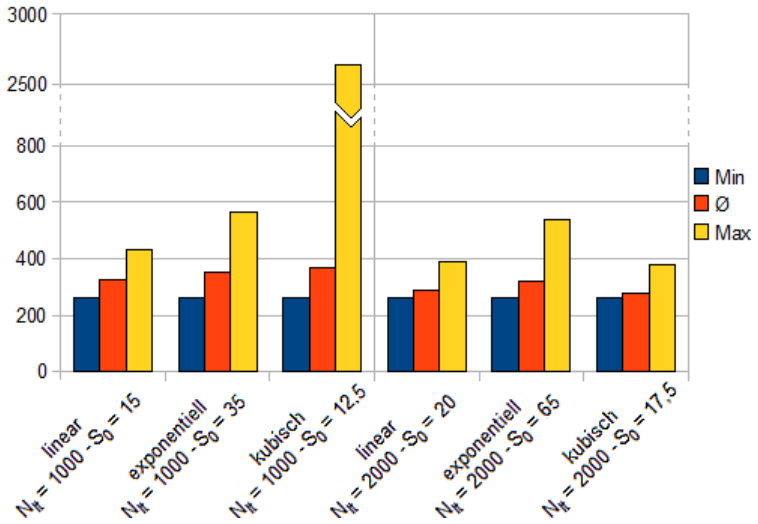

Abb. 5.34: Vergleich der Zielfunktionswerte für verschieden Abkühlungskurven mit Threshold Accepting

In Abbildung 5.34 sind die erhaltenen Zielfunktionswerte in einem Diagramm dargestellt. Es wurden jeweils die Startwerte für jede Abkühlungskurve ausgewählt, bei denen die durchschnittlichen Zielfunktionswerte am kleinsten waren. Beim linearen Absenken war das für $N_{It} = 1000$ der Startwert 15 und bei 2000 Iterationen pro Schwellenwert $S_0 = 20$. Beim exponentiellen Senken der Schwellenwerte wurde im Schnitt mit $S_0 = 35$ (bei $N_{It} = 1000$) und $S_0 = 65$ (bei $N_{It} = 2000$) die besten Ergebnisse ermittelt. Mit dem kubischen Absenken und 1000 Iterationsschritten wurde die kleinsten durchschnittlichen Zielfunktionswert bei $S_0 = 12.5$ ermittelt. Bei $N_{It} = 2000$ wurden die besten Lösungen mit $S_0 = 17.5$ beim kubischen Absenken

erreicht.

Wie dem Diagramm aus Abbildung 5.34 zu entnehmen ist, wurden mit allen Verfahren in etwas dieselben minimalen Zielfunktionswerte in 100 Versuchen ermittelt.

Mit $N_{It} = 1000$ wurden die besten Ergebnisse beim linearen Absenken mit dem Startwert 15 ermittelt. Dabei weicht der größte erhaltene Zielfunktionswert nur leicht vom Mittelwert ab. Bei den anderen beiden Methoden zum Absenken der Schwellenwerte sind die Ergebnisse bei 1000 Iterationen pro Temperatur im Schnitt schlechter. Auch sind die Schwankungen zwischen dem besten und dem schlechtesten Ergebnis größer.

Bei 2000 Iterationen pro Schwellenwert werden insgesamt bessere Ergebnisse berechnet. Die Senkungsmethoden linear und kubisch scheinen in etwa gleich gut zu sein. Beim exponentielle Senken wurden ein geringfügig größerer, durchschnittlicher Zielfunktionswert ermittelt.

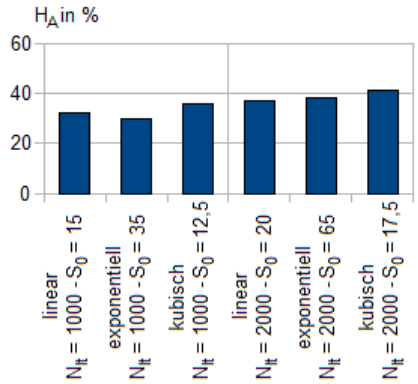

Abb. 5.35: durchschnittliche Annahmehäufigkeit für verschieden Abkühlungskurven mit Threshold Accepting

In Abbildung 5.35 ist die durchschnittliche Häufigkeit, mit der Nachbarschaftslösungen angenommen wurden, dargestellt. Es ist zu sehen, dass bei 1000 Iterationen die Verfahren, die einen etwas kleineren Anteil der Lösungen angenommen haben, besser abschnitten. Bei $N_{It} = 2000$ wurden im Schnitt über 35% der Lösungen angenommen. Dabei hebt sich die Annahmehäufigkeit des exponentiellen Absenkens der Schwellenwerte nicht von den Werten der anderen beiden Methoden ab.

In Abbildung 5.36 ist ein Diagramm zu sehen, in dem dargestellt wird, wie oft die Zeitbedingung erfüllt wurde und wie groß die durchschnittliche Überschreitung der maximal zulässigen Zeit war. Es ist zu sehen, dass sowohl bei $N_{It} = 1000$, als auch bei 2000 Iterationen pro Schwellenwert die meisten zulässigen Lösungen mit dem

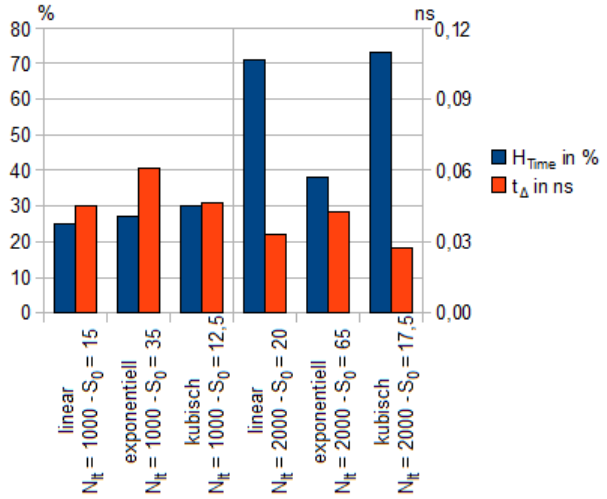

Abb. 5.36: Einhaltung der Zeitbedingung und durchschnittliche Überschreitung von $t_{max,C}$ für verschieden Abkühlungskurven mit Threshold Accepting

kubischen Absenken erreicht wurden. Bei $N_{It} = 2000$ sticht deutlich heraus, dass mit dem exponentiellen Absenken die Lösungen schlechter sind. Dort wurde die Zeitbedingung nach dem Einfügen der TSVs deutlich seltener eingehalten und auch im Schnitt stärker überschritten.
Damit kann abschließend gesagt werden, dass mit dem kubischen und dem linearen Senken der Schwellenwerte die besten Ergebnisse erzielt wurden.

5.5 Sintflutalgorithmus

5.5.1 Der Algorithmus

Der Sintflutalgorithmus [Due89] wurde ebenfalls 1989 von Dueck entwickelt. Bei diesem Verfahren ist die Annahme einer Nachbarschaftslösung nicht mehr von der Güte der aktuellen Lösung l_i abhängig. Jede Lösung, deren Zielfunktionswert einen gewissen Pegel unterschreitet, wird angenommen.
Sei $P_i \geq 0$, $i = 0, 1, 2 \ldots$ eine monoton fallende Folge von Pegelständen, für die gilt: $\forall i \geq 0 : P_i \geq P_{i+1}$. Das Annahmekriterium der Nachbarschaftslösung kann nun wie folgt beschrieben werden:

$$l_{i+1} \leftarrow l_N \in F_N(l_i) \text{ genau dann, wenn } f(l_N) \leq P_i.$$

Das bedeutet, dass alle Lösungen, die kleiner sind als der vorgegebene Pegel P_i, unabhängig von der aktuellen Lösung angenommen werden können.

Für das Absenken der Pegelstände gibt es verschiedene Herangehensweisen. Dueck schlägt in seiner Arbeit zwei Senkungsmethoden vor:

- **Prozentuales Senken der Pegel**
  Der Pegel P_i wird bei Annahme der Nachbarschaftslösung um $q\%$ des Abstand dieser zum Pegel gesenkt: $P_{i+1} = P_i - q \cdot (P_i - f(l_N))$.

- **Rekord-zu-Rekord**
  Der Pegel liegt immer um den Wert Δ über dem Zielfunktionswert der besten Lösung.

5.5.2 Ergebnisse

Im Folgenden soll untersucht werden, welche Ergebnisse mit den zwei Varianten des Sintflutalgorithmus' erzielt wurden. Dabei wurde wiederum mit zwei Längen des Algorithmus' gearbeitet: mit insgesamt 50 000 und 100 000 Iterationen. Damit sind genauso viele Iterationen ausführbar, wie bei den vorangegangenen Versuchen mit Simulated Annealing und Threshold Accepting.

Prozentuales Absenken der Pegel

Im folgenden Abschnitt werden die Ergebnisse dargestellt, die mit dem prozentualen Senken der Pegel erreicht wurden. Dabei wird – ausgehend von dem Pegel $P_0 = f(l_0)$ – der Pegel immer dann gesenkt, wenn die Nachbarschaftslösung angenommen wurde. Der Wert, um den der Pegel gesenkt wird, ist ein prozentualer Anteil der Differenz $P_i - f(l_N)$.
Die Rechenvorschrift für das Senken der Pegel um $q\%$ lässt sich wie folgt zusammenfassen:

$$P_{i+1} = \begin{cases} P_i - q \cdot (P_i - f(l_N)) & \text{, falls } l_{i+1} \leftarrow l_N \\ P_i & \text{sonst} \end{cases}$$

Anders als bei den eben untersuchten Verfahren Simulated Annealing und Threshold Accepting muss der Prozentsatz q an eine Veränderung der Iterationsanzahl angepasst werden.
In der nächsten Abbildung soll die Entwicklung der Pegel verdeutlicht werden. In 5 Versuchen wurde alle 1000 bzw. 2000 Iterationen der Pegel ausgegeben. Die

Ergebnisse wurden dann für 50 000 und 100 000 Iterationen gemittelt und die Kurven im Anschluss noch leicht geglättet.

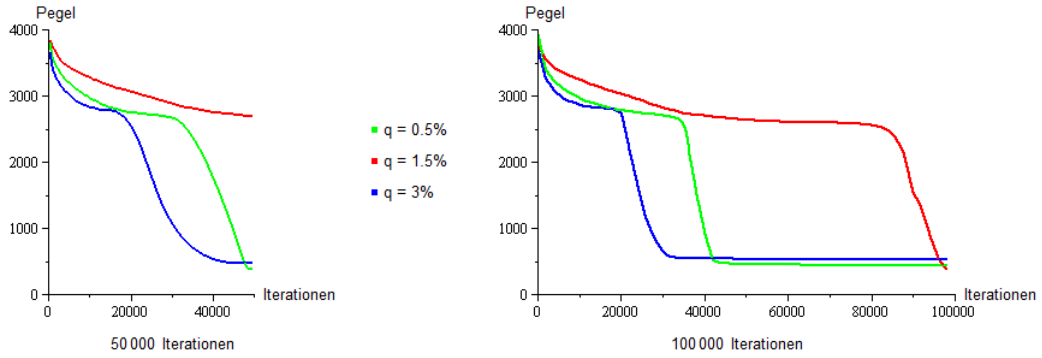

Abb. 5.37: Verlauf der Pegel mit zunehmender Iterationsanzahl beim prozentualen Senken des Pegels

In Abbildung 5.37 sind die Verläufe der Pegel für 50 000 und 100 000 Iterationen dargestellt. Beim Vergleich der beiden Diagramme fällt auf, dass die Pegel unabhängig von der Länge des Algorithmus' absinken. Deswegen wird auch mit 50 000 Iterationen bei $q = 0.5\%$ die Zeitbedingung nicht unterschritten. Der Pegel sinkt einfach zu langsam, um mit 50 000 Iterationen eine gut Lösung zu finden. Bei 100 000 Iterationen sinken die Pegel mit $q = 3\%$ ab 30 000 Iterationen kaum noch. Deswegen werden auch durch längeres Iterieren kaum bessere Lösungen gefunden.
In der folgenden Tabelle werden die Ergebnisse dargestellt. Dabei werden die durchschnittliche Häufigkeit H_A, mit der eine Nachbarschaftslösung angenommen wurde, die berechneten Zielfunktionswerte, die Kosten, die Anzahl an TSVs N_{TSV}, die Häufigkeit, mit der die Zeitbedingung nach dem Einfügen der TSVs eingehalten wurde und die Zeit, um die $t_{max,C}$ im Schnitt überschritten wurde, betrachtet.
Wie der Tabelle 5.7 zu entnehmen ist, sinkt die Annahmehäufigkeit H_A mit steigenden Prozentsatz q und steigenden Anzahl an Iterationen. Die besten Ergebnisse werden bei 50 000 Iterationen mit $q = 2\%$ erzielt. Dabei werden 20% der Nachbarschaftslösungen angenommen. Bei 100 000 Iterationen werden die besten durchschnittlichen Ergebnisse mit $q = 1\%$ ermittelt. Die Annahmehäufigkeit liegt bei diesen Werten ebenfalls bei 20%. Allerdings wurden die allerbesten Ergebnisse mit $q \geq 2\%$ erzielt. Dabei wurde die Buffer-Zeit unterschritten und damit war keine Zeitstrafe zu addieren.
In den Abbildungen 5.38 und 5.39 sind die minimalen, durchschnittlichen und maximalen Zielfunktionswerte graphisch dargestellt. Auf der Ordinate sind jeweils die

q in %	H_A ∅ in %	Zielfunktionswert ∅	Min	Max	Kosten ∅ in ct	$\varnothing N_{TSV}$	H_{Time} in %	t_Δ in ns
50 000 Iterationen								
0.5	39	2626	2543	2862	16.27	1254	0	0.096
1	28	1037	268	2872	16.14	1401	0	0.061
1.5	23	475	265	2860	16.15	1533	9	0.069
2	20	377	264	649	16.00	1505	13	0.070
2.5	18	409	263	633	16.06	1675	11	0.075
3	17	426	262	620	16.13	1755	9	0.081
100 000 Iterationen								
0.5	28	436	267	2745	15.90	1212	0	0.043
1	20	350	262	586	15.79	1365	27	0.064
1.5	16	383	261	615	15.88	1546	14	0.065
2	14	374	16	595	15.80	1560	16	0.057
2.5	13	403	17	592	15.91	1663	15	0.066
3	12	391	16	596	15.85	1637	11	0.058

Tab. 5.7: Ergebnisse des Sintflutalgorithmus' mit prozentualem Absenken des Pegels aus 100 Versuchen

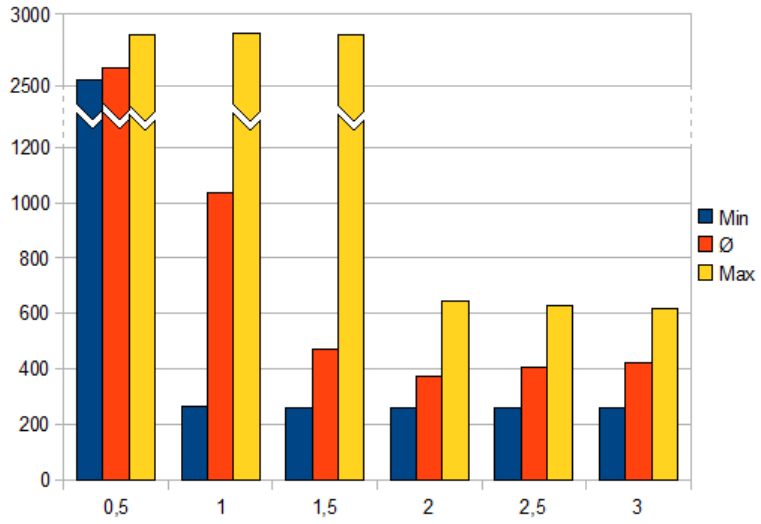

Abb. 5.38: Vergleich der berechneten Zielfunktionswerte mit dem Sintflutalgorithmus, 50 000 Iterationen und prozentualem Ansenken des Pegels

Zielfunktionswerte abgetragen, auf der Abszisse der Prozentsatz q für das Senken der Pegel. In dem Diagramm aus Abbildung 5.38 sind die Ergebnisse für 50 000 Iterationen zu sehen. Es wird deutlich, dass die Ergebnisse bei $q = 0.5\%$ praktisch unbrauchbar sind. Die Zeitbedingung wurde in der Abschätzung nicht einmal eingehalten. Das liegt daran, dass die Pegel nur sehr langsam absinken und damit mit dieser kleinen Anzahl an Iterationen keine guten Lösungen erreicht werden können. Beim Senken des Pegels nach Annahme der Nachbarschaftslösung um 1% der Differenz $P_i - f(l_N)$ fallen die Pegel immer noch zu langsam. Erst mit $q \geq 1.5$ wurden

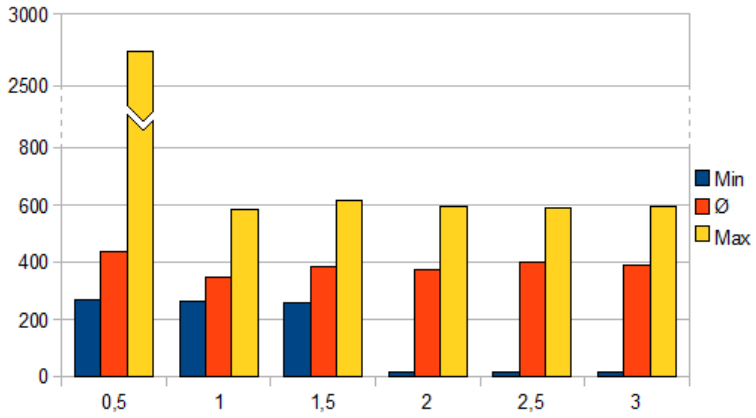

Abb. 5.39: Vergleich der berechneten Zielfunktionswerte mit dem Sintflutalgorithmus, 100 000 Iterationen und prozentualem Ansenken des Pegels

überwiegend zulässige Lösungen (bezüglich der Abschätzung ohne TSVs) ermittelt. Das ist an den deutlich kleineren durchschnittlichen Zielfunktionswerten zu sehen. Bei 100 000 Iterationen wurden auch mit kleineren Prozentwerten q gute Lösungen erzielt. Mit $q \geq 2$ wurden sogar Lösungen ermittelt, die die Buffer-Zeit unterschreiten. Es wurden aber im Schnitt kaum bessere Lösungen erzielt als bei 50 000 Iterationen. Der beste, durchschnittliche Zielfunktionswert lag bei 50 000 Iterationen bei 377 (mit $q = 2\%$). Bei 100 000 Iterationen wurde mit $q = 2\%$ ein durchschnittlicher Zielfunktionswert von 350 ermittelt. Da Verbesserung in den letzten 50 000 Iterationsschritten war also sehr klein.

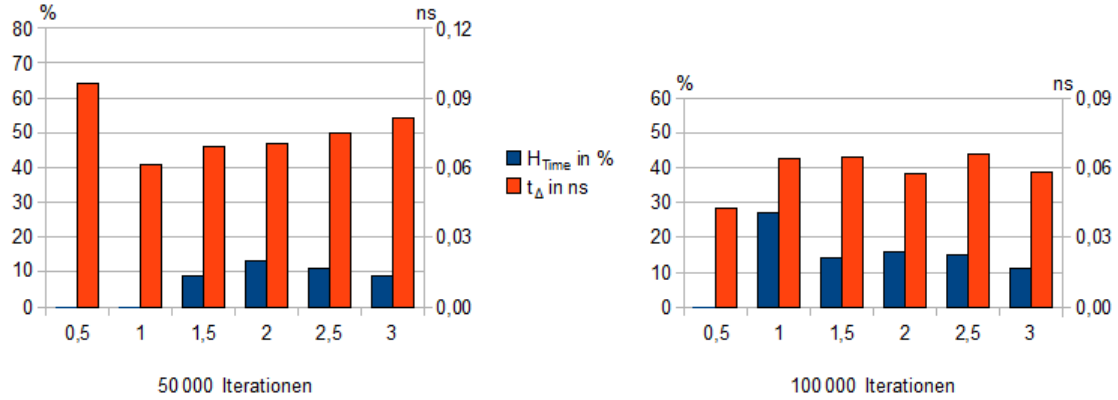

Abb. 5.40: Einhaltung der Zeitbedingung und durchschnittliche Überschreitung von $t_{max,C}$ mit dem Sintflutalgorithmus und prozentualem Absenken des Pegels

In Abbildung 5.40 sind zwei Diagramme dargestellt. Diese geben einen Überblick über die Häufigkeit H_{Time}, mit der die Zeitbedingung nach dem Einfügen der TSVs eingehalten wurde und die durchschnittliche Überschreitung von $t_{max,C}$.

In dem linken Diagramm ist zu sehen, dass die Zeitbedingung bei 50 000 Iterationen nur sehr selten eingehalten wurde. Auch die Überschreitungen waren im Schnitt sehr groß. Die meisten zulässigen Lösungen wurden bei 50 000 Iterationen mit $q = 2\%$ ermittelt, allerdings sind auch hier über 85% der Lösungen unzulässig. Bei 100 000 Iterationen wurde die Zeitbedingung deutlich öfter eingehalten. Mit 27 zulässigen Lösungen wurden die besten Ergebnisse bei $q = 1\%$ erzielt. Aber auch hier waren die meisten Lösungen unzulässig und die durchschnittlichen Überschreitungen der Zeit war sehr groß. Da auch Lösungen ermittelt wurden, die den Zeit-Buffer unterschritten haben, bei denen also ein Signal innerhalb des Chips weniger als 2.45 ns in der Abschätzung benötigt, liegt die Schlussfolgerung nahe, dass das Verfahren recht instabil ist. Sonst müssten mehr zulässige Lösungen ermittelt werden. Eine andere Ursache für die vielen unzulässigen Lösungen könnte die hohe TSV-Anzahl sein. Im Schnitt gab es 1600 Leitungen, die Superzellen auf verschiedenen Ebenen hatten – das ist so viel, dass die Fläche des Chips für die TSVs vergrößert werden muss. Deswegen ist mit hoher Wahrscheinlichkeit eine längere Leitung nötig, als in der Abschätzung ohne TSVs angenommen. Damit kann trotz reichlicher Unterschreitung von $t_{max,C}$ die Zeitbedingung nach dem Einfügen der TSVs verletzt werden.

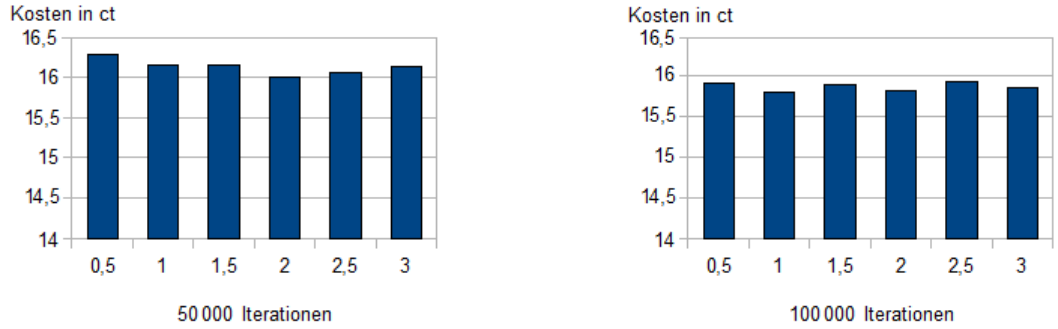

Abb. 5.41: Vergleich der durchschnittlichen Kosten pro Chip mit dem Sintflutalgorithmus und prozentualen Absenken des Pegels

In Abbildung 5.41 sind zwei Diagramme abgebildet, in denen die durchschnittlichen Kosten pro Chip graphisch dargestellt sind. Auf der Abszisse ist jeweils q abgetragen, auf der Ordinate die Kosten. Es ist zu sehen, dass die Kosten mit 50 000 Iterationen bei $q = 2\%$ am kleinsten sind. Bei 100 000 Iterationen sind die Kosten insgesamt geringer. Die besten Ergebnisse wurden mit $q = 1\%$ ermittelt. Allerdings sind Kosten auch hier recht hoch.

Rekord-zu-Rekord

Im Folgenden werden die Ergebnisse für die Variation des Sintflutalgorithmus' Rekord-zu-Rekord dargestellt. Da sich schon früh abgezeichnet hat, dass mit 50 000 und 100 000 Iterationen keine guten Ergebnisse erzielt werden, wurde die Anzahl an Iterationen auf 250 000 erhöht. Doch auch dabei wurden keine guten Lösungen ermittelt. Dies wird in der folgenden Tabelle deutlich.

Δ	H_A	Zielfunktionswert			Kosten	$\varnothing N_{TSV}$	H_{Time}	t_Δ
	$\varnothing$ in %	$\varnothing$	Min	Max	$\varnothing$ in ct		in %	in ns
50 000 Iterationen								
2.5	13	733	354	2907	16.90	2250	0	0.116
5	15	2226	271	2918	16.88	2141	0	0.111
10	20	2745	2530	2915	16.78	1950	0	0.121
20	27	2689	2549	2908	16.53	1544	0	0.119
100 000 Iterationen								
2.5	12	554	269	2888	16.88	2257	0	0.112
5	14	1189	270	2886	16.74	2050	0	0.102
10	18	2632	271	2889	16.71	1869	0	0.108
20	25	2697	2531	2925	16.60	1657	0	0.112
250 000 Iterationen								
2.5	12	550	325	680	16.92	2347	0	0.119
5	13	548	332	2773	16.76	2119	0	0.100
10	23	2643	2545	2820	16.34	1394	0	0.101
20	24	2657	2537	2880	16.41	1470	0	0.094

Tab. 5.8: Ergebnisse des Sintflutalgorithmus' mit Rekord-zu-Rekord aus 100 Versuche

In Tabelle 5.8 sind die Häufigkeit H_A, mit der eine Nachbarschaftslösung angenommen wurde, die berechneten Zielfunktionswerte, die Kosten und die TSV-Anzahl N_{TSV} eingetragen. Außerdem werden noch Angaben über die Häufigkeit H_{Time}, mit der Lösungen nach dem Einfügen der TSVs zulässig sind und die durchschnittliche Überschreitung von $t_{max,C}$ gemacht.

Beim Betrachten der vorletzten Spalte, in der notiert ist, wie viele der Lösungen nach dem Einfügen der TSVs die Zeitbedingung einhalten, fällt auf, dass keine einzige zulässige Lösung erzeugt wurde. Damit ist dieser Algorithmus absolut unbrauchbar. Auch die anderen Werte in der Tabelle sind sehr schlecht. Die Zielfunktionswerte liegen bei $\Delta \geq 10$ fast immer über 2500, es wurde also so gut wie keine Lösung gefunden, in der wenigstens in der Abschätzung ohne TSVs die Zeitbedingung eingehalten wurde. Bei $\Delta \leq 10$ wurde die Zeitbedingung in der Abschätzung zwar häufig eingehalten, trotzdem sind die Lösungen praktisch unbrauchbar. Es sind viel zu viele TSVs einzufügen, was durch eine starke Vergrößerung der Fläche und die Defektwahrscheinlichkeit der TSVs zu sehr hohen Kosten führt. Deswegen sind

auch die Endlösungen alle unzulässig, da durch die TSVs große Umwege bei der Verdrahtung gemacht werden müssen.
Die Ursache für diese schlechten Ergebnisse liegt vermutlich darin, dass bei kleinem Δ kaum die Möglichkeit besteht, lokale Minima zu überwinden. Nach Erhöhen von Δ findet keine Steuerung des Algorithmus statt und der Pegel sinkt praktisch nicht ab.

5.6 Vergleich der Verfahren

In diesem Abschnitt sollen die Verfahren insgesamt verglichen werden. Da der Sintflutalgorithmus mit Rekord-zu-Rekord so schlecht abschnitt, wird dieser hier nicht betrachtet.
Um zu sehen, wie die Verfahren nach einer weiteren Steigerung der Iterationsanzahl abschneiden, wurden mit den verschiedenen Methoden 50 Versuche mit $N_{It} = 5000$ bzw. 250 000 Iterationen ausgeführt. Dabei wurden die Startwerte gewählt, die sich in den vorangegangenen Berechnungen als günstig erwiesen haben.

Methode	Zielfunktionswert			Kosten	$\varnothing N_{TSV}$	H_{Time}	t_Δ
	$\varnothing$	Min	Max	$\varnothing$ in ct		in %	in ns
Simulated Annealing – $N_{It} = 5000$							
lin – $T_0 = 20$	269	259	362	15.15	953	90	0.047
exp – $T_0 = 25$	285	15	383	15.24	1064	68	0.035
log – $T_0 = 30$	263	259	373	15.15	919	92	0.013
Threshold Accepting – $N_{It} = 5000$							
lin – $S_0 = 25$	267	260	352	15.13	940	92	0.025
exp – $S_0 = 65$	284	15	375	15.2	1026	74	0.031
kub – $S_0 = 20$	270	260	364	15.15	948	90	0.027
Sintflutalgorithmus – 250 000 Iterationen							
$q = 0.5\%$	329	261	459	15.57	1259	38	0.048
$q = 1\%$	246	15	573	15.55	1329	40	0.045

Tab. 5.9: Ergebnisse mit den verschiedenen Verfahren aus 50 Versuche

In der Tabelle 5.9 sind für die drei Verfahren Simulated Annealing, Threshold Accepting und Sintflutalgorithmus mit prozentualem Absenken der Pegel die Ergebnisse mit insgesamt 250 000 Iterationen aus 50 Versuchen dargestellt. Eingetragen sind die ermittelten Zielfunktionswerte, die durchschnittlichen Kosten pro Chip, die TSV-Anzahl N_{TSV}, sowie die Häufigkeit H_{Time}, mit der die Lösung nach dem Einfügen der TSVs zulässig ist und die durchschnittliche Überschreitung t_Δ der maximal zulässigen

Verweildauer eines Signals innerhalb des Chips. Die Werte wurden beim Simulierten Abkühlen und bei Threshold Accepting für alle drei vorgestellten Methoden zum Absenken der Temperatur bzw. der Schwellenwerte ermittelt. Der Sintflutalgorithmus wurde mit zwei Prozentsätzen q zum Absenken der Pegel ausgeführt.
Aus Tabelle 5.9 ist zu entnehmen, dass beim Simulated Annealing die besten Ergebnisse mit dem logarithmischen Absenken der Temperatur erzielt wurden. Das lineare Absenken schneidet fast genauso gut ab. Mit dem exponentielle Absenken der Temperatur wurde mit einem Zielfunktionswert von 15 die beste Lösung ermittelt. Allerdings sind die Zielfunktionswerte im Schnitt größer, als bei den anderen beiden Senkungsmethoden. Diese Variante des Simulated Annealing ist also instabiler.
Beim Threshold Accepting zeichnet sich ein ähnlich klares Bild ab. Die besten Werte wurden mit dem linearen Absenken ermittelt. Das exponentielle Senken der Schwellenwerte führte auch hier zu sehr unterschiedlichen Lösungen. Mit dem kubischen Verlauf der Schwellenwerte wurden nur geringfügig schlechtere Lösungen als beim linearen Absenken ermittelt.
Die besten durchschnittlichen Zielfunktionswerte aus allen Verfahren werden aber mit dem Sintflutalgorithmus erzeugt. Hierbei muss allerdings negativ angemerkt werden, dass ein deutlich größerer Anteil an Endlösungen nach dem Einfügen der TSVs unzulässig ist. Auch schwankt die Güte der Lösung stärker als bei den anderen beiden Verfahren. Deswegen ist diese Methode nur dann zu empfehlen, wenn mehrere Berechnungen ausgeführt werden und mit das besten Ergebnis weiterverarbeitet wird.
Die Verfahren Simulated Annealing und Threshold Accepting schneiden bei 5000 Iterationen pro Temperatur bzw. Schwellenwert ungefähr gleich gut ab. Da Threshold Accepting nicht mit logarithmischen Absenken der Schwellenwerte implementiert wurde, ist schwer zu sagen, ob dabei bessere Ergebnisse als beim Simulierten Abkühlen erzeugt werden. Da aber die Ergebnisse bei den identischen Kurven zum Absenken der Schwellenwerte bzw. Temperatur (linear und exponentiell) bei Threshold Accepting besser waren, ist es naheliegend, dass auch beim logarithmischen Absenken der Schwellenwerte bessere Ergebnisse erzielt werden.
Beim Vergleich der Ergebnisse aus den Berechnungen mit 1000 und 2000 Iterationen pro Schwellenwert bzw. Temperatur fällt auf, dass Threshold Accepting dabei ebenfalls besser abschneidet als Simulated Annealing. Um diese Tatsache darzulegen,

sind in der folgenden Tabelle die besten Ergebnisse von Simulated Annealing und Threshold Accepting bei $N_{It} = 1000$ und $N_{It} = 2000$ noch einmal dargestellt.

Methode	Zielfunktionswert			Kosten	$\varnothing N_{TSV}$	H_{Time}	t_Δ
	$\varnothing$	Min	Max	$\varnothing$ in ct		in %	in ns
Simulated Annealing – $N_{It} = 1000$							
lin – $T_0 = 10$	332	264	508	15.66	1281	27	0.049
exp – $T_0 = 35$	365	263	577	15.78	1459	16	0.057
log – $T_0 = 10$	416	263	2708	15.98	1607	7	0.064
Threshold Accepting – $N_{It} = 1000$							
lin – $S_0 = 15$	328	264	432	15.63	1263	25	0.045
exp – $S_0 = 35$	355	263	563	15.76	1397	27	0.061
kub – $S_0 = 12.5$	370	264	2631	15.67	1242	30	0.046
Simulated Annealing – $N_{It} = 2000$							
lin – $T_0 = 15$	293	262	388	15.36	1075	66	0.037
exp – $T_0 = 85$	326	262	564	15.51	1253	42	0.044
log – $T_0 = 20$	296	262	386	15.44	1098	56	0.045
Threshold Accepting – $N_{It} = 2000$							
lin – $S_0 = 20$	287	262	392	15.33	1040	71	0.033
exp – $S_0 = 65$	320	262	538	15.51	1236	38	0.043
kub – $S_0 = 17.5$	278	262	380	15.28	983	73	0.027

Tab. 5.10: Ergebnisse mit den verschiedenen Verfahren aus 100 Versuche

In Tabelle 5.10 sind die jeweils besten Ergebnisse mit Simulated Annealing und Threshold Accepting dargestellt, um vergleichen zu können, welches Verfahren bei kleinerer Iterationsanzahl bessere Resultate lieferte. Dabei fällt auf, dass sowohl bei 1000, als auch bei 2000 Iterationen pro Schwellenwert bzw. Temperatur mit Threshold Accepting im Allgemeinen bessere Ergebnisse ermittelt werden. Mit $N_{It} = 1000$ liegen mit Simulated Annealing die besten durchschnittlichen Zielfunktionswerte bei 332. Mit Threshold Accepting und linearem Absenken wurde im Schnitt ein Zielfunktionswert von 328 erzielt. Auch die durchschnittlichen Kosten pro Chip sind mit 15.63 ct kleiner.

Auch bei 2000 Iterationen pro Schwellenwert bzw. Temperatur sind die Resultate mit Threshold Accepting deutlich besser. Die Zielfunktionswerte sind im Schnitt kleiner und die Anzahl an zulässigen Lösungen ist deutlich höher.

Damit wird deutlich, dass mit Threshold Accepting im Allgemeinen die besten Lösungen berechnet werden. Lediglich bei einer hohen Anzahl an Iterationen sollte das mehrfache Ausführen des Sintflutalgorithmus' mit prozentualem Senken der Pegel bevorzugt werden.

Möglicherweise könnte eine Kombination der Verfahren Simulated Annealing und Threshold Accepting noch bessere Ergebnisse liefern. Es wäre denkbar, eine Nachbarschaftslösung $l_N \in F_N(l_i)$ mit folgender Wahrscheinlichkeit anzunehmen:

$$P(l_{i+1} \leftarrow l_N \in F_N(l_i)) = \min\{1, e^{\frac{f(l_i)-(f(l_N)-S_i)}{T_i}}\}.$$

Die Nachbarschaftslösung wird dann mit Sicherheit angenommen, wenn deren Zielfunktionswert höchsten im den Wert S_i größer ist als die Güte der aktuellen Lösung, andernfalls nur mit einer temperaturabhängigen Wahrscheinlichkeit. Damit wären die Vorteile beider Verfahren kombiniert. Kleine Verschlechterungen werden immer angenommen, was dem Algorithmus die Möglichkeit gibt, schnell aus lokalen Minima zu entkommen. Aber auch große Verschlechterungen können mit einer gewissen Wahrscheinlichkeit angenommen werden, um auch am Ende lokale Minima überwinden zu können.

6 Zusammenfassung und Ausblick

In Rahmen der Diplomarbeit ist es gelungen, ein hierarchisches Programm zu entwickeln, welches einen elektrischen Schaltkreis auf mehrere Ebenen aufteilt und damit einen 3D-Aufbau eines Chips ermöglicht.
Mit dem Partitionierungsverfahren hMETIS wurde der Hypergraph des elektrischen Schaltkreises vereinfacht, indem die Zellen einer Knotenteilmenge der Partition zu Superzellen fusioniert worden. Damit wurde das Problem, mehrere Hunderttausend Standardzellen auf mehreren Ebenen zu platzieren, auf die Positionierung von wenigen großen Superzellen vereinfacht.
Die Platzierung der Superzellen wurde mit der Datenstruktur Sequenz-Paar und einem heuristischen Optimierungsverfahren ausgeführt. Es wurde verglichen, welches der vorgestellten Such-Verfahren die besten Ergebnisse bei der Platzierung ermittelt. Dabei hat sich herausgestellt, das mit Threshold Accepting im Allgemeinen die besten Ergebnisse erzielt wurden.

Im Weiteren sollte untersucht werden, ob bei anderen Schaltungen ähnliche Ergebnisse ermittelt werden. Auch ist die Schnittstelle zu den kommerziellen P&R-Tool zu erweitern. Dabei ist insbesondere zu testen, ob das verwendeten Modell für die Zeit, die ein Signal innerhalb des Chips zwischen zwei Registerzellen benötigt, realistische Ergebnisse liefert.
Auch sollte untersucht werden, inwiefern durch eine Nachbesserung mit mehr Superzellen (durch Teilen der mit hMETIS erzeugten Superzellen) die Ergebnisse der Platzierung weiter verbessert werden können. Dies könnte vor allem dann wichtig sein, wenn weniger TSVs auf einer Ebenen platziert werden können, als in dieser Arbeit angenommen wurde.
Im Rahmen der Arbeit ist auch aufgefallen, dass hMETIS – beim mehrfachen Teilen eines Hypergraphen in 2^j Blöcke – sehr unterschiedliche Ergebnisse liefert.

Möglicherweise könnte mit einem anderen Verfahren zur Partitionierung oder einer komplexen Strategie für das Zusammenfassen von Standart-Zellen zu Superzellen schneller eine gute Vereinfachung des Schaltkreises gefunden werden.
In dieser Arbeit blieb ein wichtiger Punkt in der Entwicklung des Layouts eines Chips unberücksichtigt: die Thermik. Da alle Zellen Wärme abgeben und auch durch den elektrischen Widerstand und die kapazitiven Effekte in den Leitungen elektrische Energie in Wärme umgewandelt wird, kann es zu Überhitzungen kommen. Deswegen sollte bei der Platzierung der Zellen darauf geachtet werden, dass Zellen, die viel Wärme abgeben, nicht zu dicht beieinander platziert werden. Dieser wichtige Aspekt der Layout-Entwicklung sollte in das Programm eingebaut werden.

Literaturverzeichnis

[AK90] Aarts, Emile H. L.; Korst, Jan: *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing.* John Wiley & Sons, Chichester 1990

[AL03] Aarts, Emile H. L.; Lenstra, Jan Karel [Hrsg.]: *local search in combinatorial optimization.* Princeton University Press, Princeton 2003.

[Alb07] Albers, Jan: *Grundlagen integrierter Schaltungen - Bauelemente und Mikrostrukturierung.* Carl Hanser Verlag, München 2007.

[ACG+99] Ausiello, Giorgio; Crescenzi, Pierluigi; Gambosi, Giorgio; Kann, Viggo; Marchetti-Spaccamela, Alberto; Protasi, Marco: *Combinatorial Optimization Problems and Their Approximability Properties.* Springer, Berlin 1999.

[Aze92] Azencott, Robert [Hrsg.]: *Simulated Annealing - Parallelization Techniques.* John Wiley & Sons, New York 1992.

[Ber89] Berge, Claude: *Hypergraphs.* In: *North-Holland Mathemtical Library Vol. 45.* North-Holland, Amsterdam 1989.

[Due89] Dueck, Gunter: *New optimization heuristics - the great deluge algorithm and the record-to-record travel.* In: *Technical reports / IBM Germany, Wissenschaftliches Zentrum Heidelberg* IBM Deutschland GmbH, Heidelberg 1989.

[DS90] Dueck, Gunter; Scheuer, Tobias: *Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing.* In: *Journal of Computational Physics 90*, S. 161-175. Academic Press, London 1990.

[DW89] Dueck, Gunter; Wirsching, Jens: *Threshold accepting algorithms for multi-constraint 0-1 Knapsack problems.* In: *Technical reports / IBM Germany, Wissenschaftliches Zentrum Heidelberg* IBM Deutschland GmbH, Heidelberg 1989.

[DX09] Dong, Xiangyu; Xie, Yuan: *System-Level Cost Analysis and Design Exploration für Three-Dimensional Integrated Chirciuts (3D ICs)*. Asia and South Pacific Design Automation Conference 2009.

[FL09] Fischbach, Robert; Lienig, Jens; Hertwig, Jörg: *Modern 3D Data Structures: Classification, Comparison and Solution Space Investigation*. TU Dresden, Fakultät für Elektrotechnik und Informationstechnik 2009.

[FM82] Fiduccia, C.M.; Mattheyses, R.M.: *A Linear-Time Heuristic for Improving Network Partitions*. 19th Design Automation Conference S.175-181, 1982.

[GCY99] Guo, Pei-Ning; Cheng, Chung-Kuan; Yoshimura, Takeshi: *An O-Tree Representation of Non-Slicing Floorplan and Its Applications*. New Orleans 1999.

[Her02] Hermann, Paul: *Rechnerarchitektur: Aufbau, Organisation und Implementierung, inklusive 64-Bit-Technologie und Parallelrechner*. Vieweg, Wiesbaden 2002.

[HHC+00] Hong, Xianlong; Huang, Gang; Cai, Yici; Gu, Jiangchun; Dong, Sheqin; Cheng, Chung-Kuan; Gu, Jun: *Corner Block List: An Effective and Efficient Topological Representation of Non-Slicing Floorplan*. International Conference on Computer-Aided Design 2000.

[HJR09] Hwang, Myeong-Eun; Jung, Seong-Ook; Roy, Kaushik: *Slope Interconnect Effort: Gate-Interconnect Interdependent Delay Modeling for Early CMOS Circuit Simulation*. In: *IEEE Transactions on circuits and systems - I: Regular Papers, Vol. 56, No. 7, July 2009*. IEEE, New York 2009.

[KAKS99] Karypis, George; Aggarwal, Rajat; Kumar, Vipin; Shekhar, Shashi: *Multilevel Hypergraph Partitioning: Applications in VLSI Domain*. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 7, Nr. 1, März 1999*. S. 69-79. IEEE, New York 1999.

[KAL09] Kim, Dea Hyun; Athikulwongse, Krit; Lim, Sung Kyu: *A Study of Through-Silicon-Via Impact on the 3D Stacked IC Layout*. ICCAD'09, November 02–05, San Jose (USA) 2009.

[Kra94] Kraus, Peter: *Analytische Platzierungsverfahren für den makrozellbasierten IC-Entwurf*. In: *Fortschritt-Berichte VDI, Reihe 20, Nr. 116*. VDI-Verlag, Düsseldorf 1994.

[Lau10] Lau, John H.: *TSV Manufacturing Yield and Hidden Costs for 3D IC Integration*. 2010 Electronic Components and Technologie Conference.

[MFNK96] Murata, Horoshi; Fujiyoshi, Kunihiro; Nakatake, Shigetoshi; Kajitani, Yoji: *VLSI Module Placement Based on Rectangle-Packing by the Sequenz-Pair*. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, Nr. 12, Dezember 1996.* S. 1518-1524. IEEE, New York 1999.

[Mis00] Miskowiec, Peter: *Schaltungsbezogene Modellierung der Ausbeute und des Ausfallrisikos mikroelektronischer Schaltkreise unter Berücksichtigung defektinduzierter Ausfallmechanismen*. Dissertation. Gerhard-Mercator-Universität-Gesamthochschule Duisburg, 2000.

[Mös92] Möschwitzer, Albrecht: *Grundlagen der Halbleiter- & Mikroelektronik - Band 2: Integrierte Schaltkreise*. Carl Hanser Verlag, München 1992.

[Sch10] Schwope, Andreas: *ASICs*. `http://www.andreas-schwope.de/ASIC_s/` [Stand: 20.06.2010]

[Sch95] Schmeck, Hartmut: *Analyse von VLSI-Algorithmen*. Spektrum, Akademischer Verlag, Heidelberg 1995.

[Unb09] Unbekannter Autor: *ASIC-System On Chip (SoC) - VLSI Design*. `http://asic-soc.blogspot.com/2009/06/timing-paths.html` [Stand: 15.11.2010]

[YSNK00] Yamazaki, Hiroyuki; Sakanushi, Keishi; Nakatake, Shigetoshi; Kajitani, Yoji: *The 3D-Packing by Meta Data Structure and Packing Heuristik*. In: *IEICE Trans. Fundamentals, Vol. E83-A, No. 4*. 2000.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.
Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Dresden, 15. Dezember 2010

Franziska Heinicke