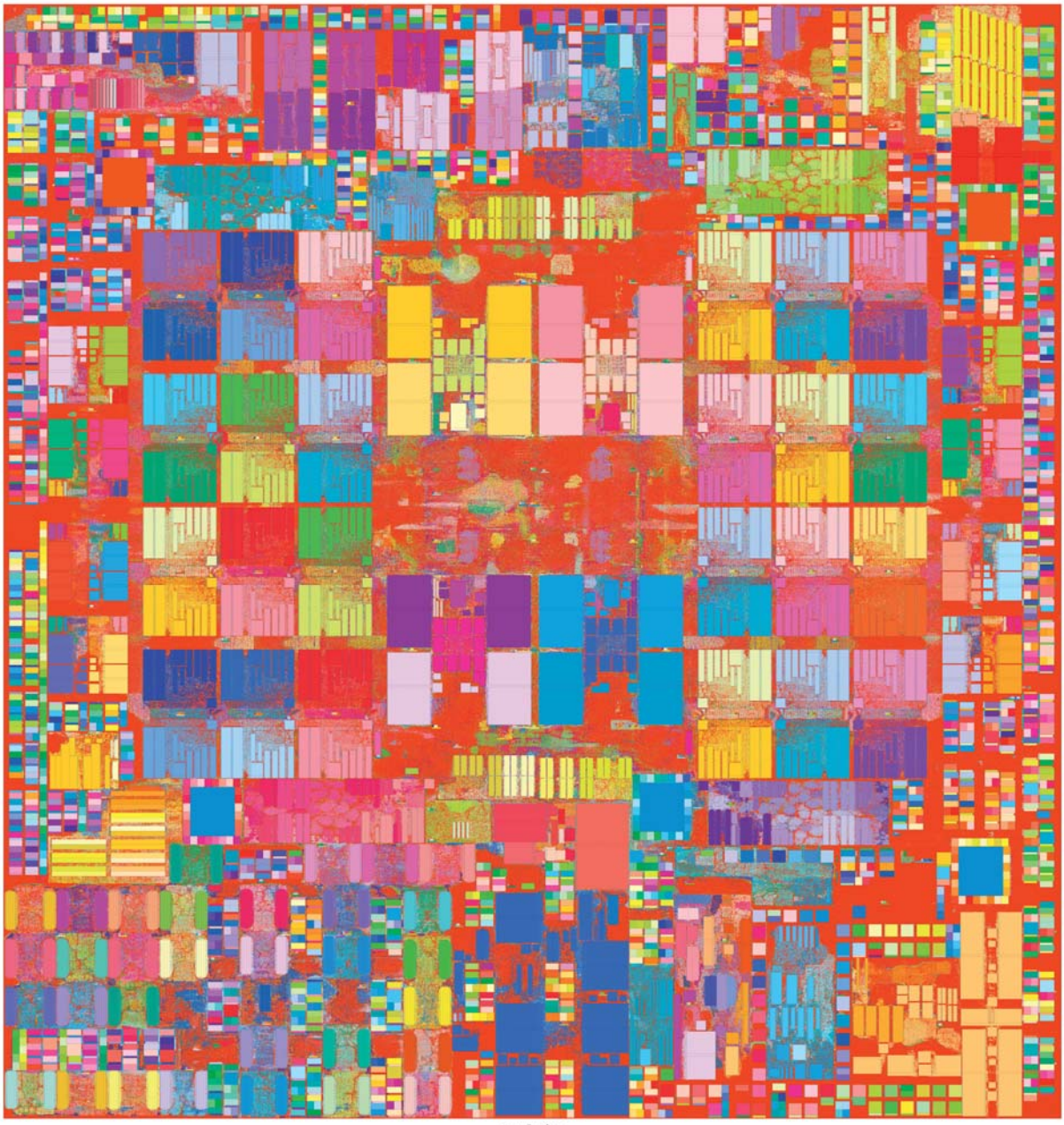
Chip-Design und Mathematik

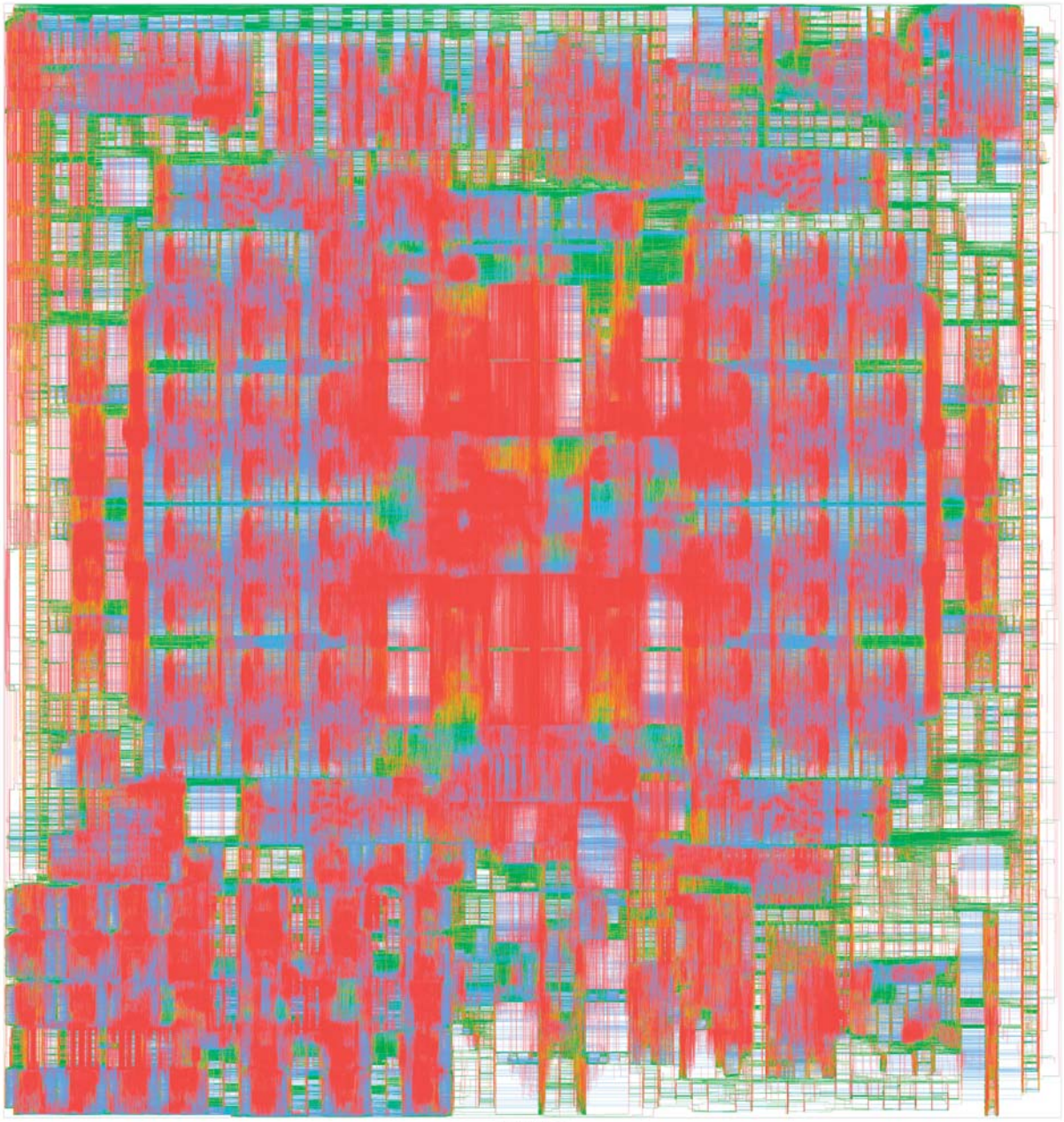
Jens Vygen

1 Einführung

Die technologische Entwicklung von Halbleiterschaltkreisen, kurz Chips, sucht ihresgleichen. Noch immer gehorcht sie weitgehend dem „Mooreschen Gesetz“, nach dem sich die Anzahl der Transistoren auf einem Chip mit jeder technologischen Generation, also zur Zeit etwa alle zwei Jahre, verdoppelt.[1] Während die ersten Schaltkreise, bis in die 1980er Jahre hinein, weitgehend manuell entworfen wurden, dominieren seit über 20 Jahren automatische Entwurfswerkzeuge, manchmal *electronic design automation tools* genannt.



Platzierung des Chips „Ludwig“ mit rund acht Millionen Bauteilen (vgl. Ausschnitt auf S. 15)

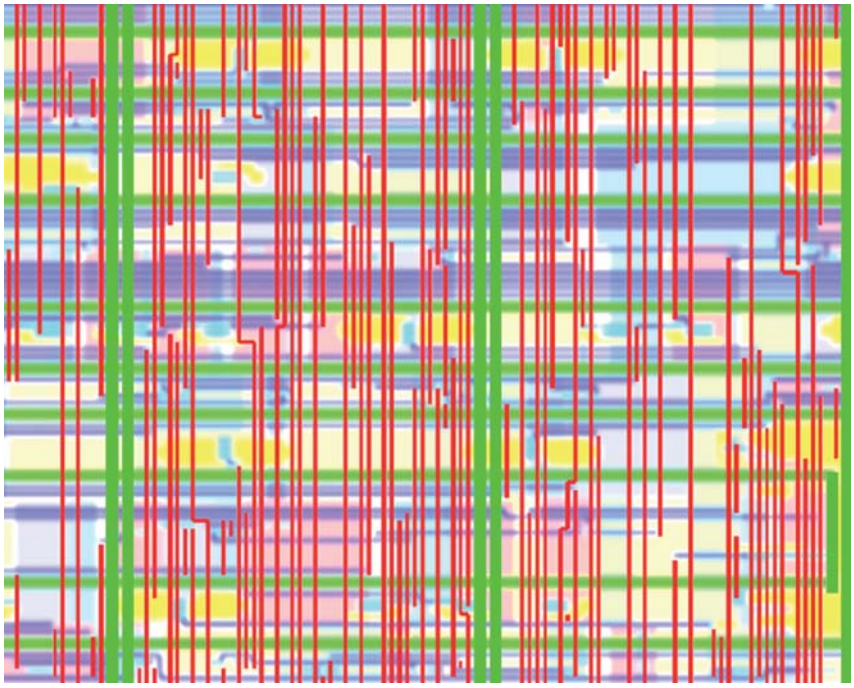


Verdrahtung des Chips „Ludwig“ mit rund einem Kilometer Draht

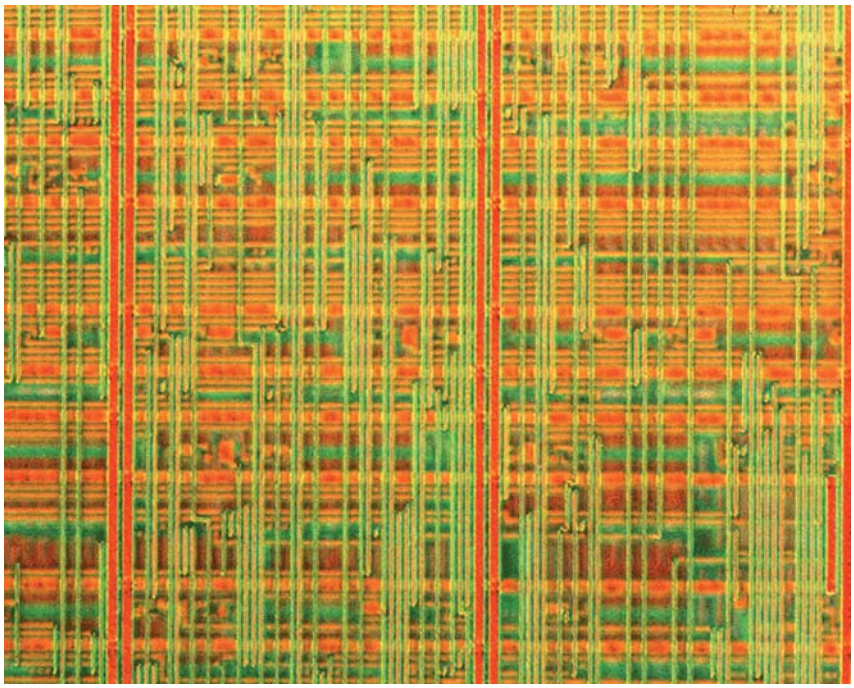
Die ersten dieser Programme waren noch ziemlich primitiv und enthielten recht einfache Heuristiken. Erst nach und nach hat die Mathematik Einzug gehalten. Tatsächlich enthält das Chip-Design eine Vielzahl interessanter – und schwieriger – mathematischer Probleme (vgl. [5]). Auch wenn das Chip-Design in einer sehr großen Zahl von Publikationen als Motivation für das bearbeitete Problem herangezogen wurde, kamen nur wenige der theoretischen Fortschritte tatsächlich zur Anwendung. Meist, weil die Problemstellung eben doch nicht der Anwendung gerecht wurde, manchmal auch, weil es eben irgendwie auch ohne Mathematik ging. Kurz gesagt, die Kluft zwischen Ingenieuren und Mathematikern war zu groß.

Seit Mitte der 1980er Jahre ist Chip-Design ein Forschungsschwerpunkt des von Bernhard Korte gegründeten Forschungsinstituts für Diskrete Mathematik. Dank intensiver Kooperationen mit der Industrie, insbesondere IBM, bestand hier von Anfang an der direkte Kontakt zwischen Mathematik und Praxis. Heute haben mehrere Chip-Designer der IBM ihre Büros direkt in unserem Institut. Dies stellt sicher, dass der Anwendungsbezug der bearbeiteten Probleme nie aus den Augen verloren wird.

Eine stetige Herausforderung stellt allerdings dar, dass das gesamte Spektrum von mathematischer Grundlagenforschung über die Entwicklung neuer Algorithmen bis



Ausschnitt aus dem Design



Derselbe Ausschnitt von einem Foto des produzierten Chips

hin zu fertiger Software abgedeckt werden muss, um wirklich den größtmöglichen Nutzen aus dieser Konstellation zu ziehen. Die theoretischen Ergebnisse werden veröffentlicht; die praktischen Ergebnisse sind unter dem Namen *BonnTools* [6] weltweit in Design-Zentren im Einsatz.

Die BonnTools enthalten viel Mathematik, aber natürlich nicht nur Mathematik. Sie bestehen aus einigen neu entwickelten Algorithmen für Kernprobleme, daneben einer großen Bibliothek mit Datenstrukturen und Algorithmen für Standardprobleme der Diskreten Optimierung und Algorithmischen Geometrie, sowie auch einigen heuristischen Vor- und Nachbearbeitungsverfahren.

Viele der komplexesten Chips von führenden Technologiefirmen sind mit den BonnTools entworfen worden. Sie wären ohne diese Programme, und das heißt letztlich

ohne die darin steckende Mathematik, nicht realisierbar gewesen, oder nur mit wesentlich geringerer Leistung. Die großen Bilder (Abbildungen 1 und 2) zeigen ein aktuelles Beispiel, einen Telekommunikationschip, den wir Ludwig getauft haben, und der ab Mitte 2009 in 65-nm-Technologie[2] produziert werden wird.

Die Bilder zeigen die Platzierung der rund acht Millionen rechteckigen Bauteile, die jeweils eine logische Funktion implementieren oder Daten speichern, und deren Verdrahtung. Dies sind nach wie vor zwei der Kernprobleme des Chip-Designs: die Bauteile müssen überlappungsfrei auf der gegebenen Fläche platziert und anschließend so miteinander verbunden werden, dass alle Nebenbedingungen eingehalten werden. Wir konzentrieren uns hier auf das zweite Problem, die Verdrahtung, wollen aber nicht unerwähnt lassen, dass mit Timing-Optimierung, Design von Netzwerken für Taktsignale, und Logik-Synthese viele weitere wichtige und interessante Problemkreise existieren, zu denen es ebenso viel zu sagen gäbe. Die interessierten Leser seien hier auf [1] und [4] verwiesen.

Eine immer wieder faszinierende Eigenschaft des Chip-Designs ist, dass die Daten und Nebenbedingungen bis auf Nanometer und Pikosekunde genau vorliegen, und eine im mathematischen Modell berechnete Lösung dann auch – abgesehen von unvermeidlichen Fertigungstoleranzen – exakt so gebaut wird. Wir können von uns entworfene Chips unter dem Mikroskop betrachten und sehen genau die von unseren Algorithmen berechneten Strukturen (Abbildung 3). Allenfalls über die Zielfunktion wird mitunter diskutiert, und darüber, welche Anforderungen relaxiert werden sollen, wenn sich eine Instanz als unlösbar erweist (was in der Praxis häufig vorkommt).

2 Verdrahtung

Ein Chip enthält Millionen kleine und einige größere Bauteile (Abbildung 1), die alle jeweils mehrere Anschlusspunkte (*Pins*) haben, über die sie Eingabesignale empfangen und Ausgabesignale weitergeben. Diese Pins müssen mit leitenden Verbindungen (in der Regel sehr dünne Kupferdrähte) korrekt verbunden werden. Genauer ist die Menge der Pins in Teilmengen zerlegt, die *Netze* genannt werden. Jedes Netz besteht aus einem Ausgangspin eines Bauteils und Eingangspins anderer Bauteile, die das Signal weiterverarbeiten. Daher müssen die Pins jedes Netzes verbunden werden.

Für die Drähte stehen einige Verdrahtungsebenen zur Verfügung, in denen die Drähte nur horizontal oder vertikal verlaufen dürfen.[3] Die Drähte müssen eine bestimmte Mindestbreite haben. Übereinander liegende Drähte in benachbarten Verdrahtungsebenen können durch so genannte *Vias* miteinander verbunden werden. Zu verschiedenen Netzen gehörende Drähte und Vias müssen hinreichenden Abstand voneinander haben.

Abbildung 2 zeigt die Verdrahtung des Chips Ludwig. Auch wenn man trotz der rund zehnfachen Vergrößerung keine einzelnen Drähte erkennen kann, bekommt man doch einen gewissen Eindruck. Die Farben entsprechen hier den Verdrahtungsebenen: Drähte auf der obersten Ebene (fast ausschließlich vertikal) sind rot, Drähte auf der zweitobersten Ebene (horizontal) sind blau. Dann folgen grün, orange, und weitere kaum noch erkennbare (weil darunterliegende) Ebenen. Die Gesamtlänge aller Drähte beträgt hier circa einen Kilometer.

Die genauen Parameter für Mindestbreiten und Mindestabstände sollen uns hier nicht kümmern. Es verfälscht das Problem nicht zu sehr, anzunehmen, dass alle diese Zahlen identisch sind. Dann bietet es sich an, alle Drähte an einem zweidimensionalen äquidistanten Gitter auszurichten, etwa nur ganzzahlige Koordinaten zuzulassen.[4] Also erhalten wir folgende Problemformulierung:

Instanz: Ein dreidimensionaler Gittergraph G, d. h. ein endlicher Subgraph von $(\mathbb{Z}^3, \{\{v, w\} : v, w \in \mathbb{Z}^3, \|v - w\|_1 = 1\})$. Eine Menge von paarweise disjunkten Netzen, wobei jedes Netz eine Teilmenge der Knotenmenge $V(G)$ ist.

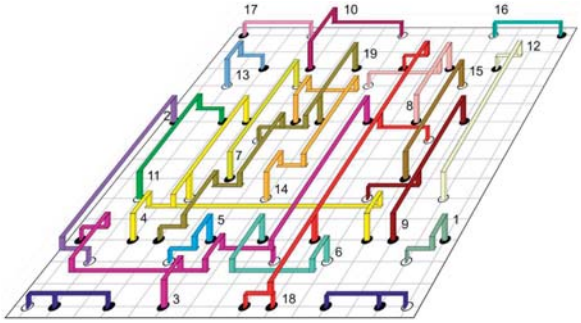
Aufgabe: Finde einen *Steinerbaum* T_N für jedes Netz N, d. h. einen minimalen zusammenhängenden Subgraphen von G, der alle Knoten aus N enthält, so dass keine zwei dieser Steinerbäume einen Knoten gemeinsam haben.

Abbildung 4 zeigt ein kleines, sehr einfaches Beispiel mit knotendisjunkten Steinerbäumen für 21 Netze. In diesem Beispiel gibt es nur drei Ebenen, von denen die unterste nur die Pins enthält.

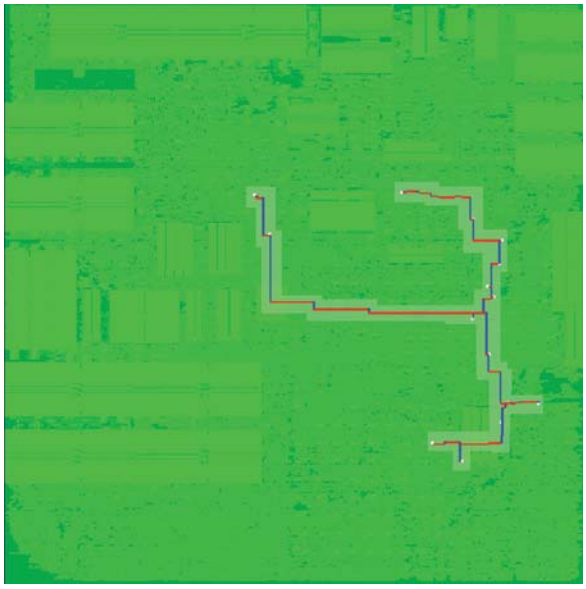
Natürlich ist dieses Problem, das auch als knotendisjunktes Packen von Steinerbäumen bezeichnet wird, nicht identisch mit dem Verdrahtungsproblem. Dennoch ist es eine erste brauchbare Näherung. Sollte jemand einen guten Algorithmus für das obige Problem finden, ließe sich dieser sicherlich (in Kombination mit anderen Verfahren) gewinnbringend zur Verdrahtung einsetzen.

Allerdings ist das recht unwahrscheinlich, denn es scheint sich um ein äußerst schwieriges Problem zu handeln: es ist nicht nur *NP*-schwer im Sinne der Komplexitätstheorie, sondern es sind auch kaum positive theoretische Ergebnisse oder größere optimal gelöste Instanzen bekannt.

Zudem sind die in der Praxis auftretenden Instanzen wirklich riesig: Man sucht zum Beispiel 10 Millionen knotendisjunkte Steinerbäume in einem Graphen mit rund 10^{11} Knoten. Schließlich sucht man natürlich nicht irgendwelche Steinerbäume, sondern es sollen Nebenbedingungen und Zielfunktionen wie Signalübertragungszeiten, Stromverbrauch, und erwartete Fehlerquote in der Herstellung eingehalten bzw. optimiert werden. Als erste Näherung kann man sagen: die Steinerbäume sollen möglichst kurz sein.



Einfaches Beispiel für 21 knotendisjunkte Steinerbäume in zwei Ebenen



Global-Routing-Korridor für ein Netz und der Steinerbaum für dieses Netz

Das hört sich ziemlich hoffnungslos an, und es besteht in der Tat auch kaum Aussicht auf einen Algorithmus, der jede solche Instanz lösen könnte oder feststellen könnte, dass sie unlösbar ist. Was kann der Mathematiker also hier beitragen?

3 Verdrahtung ohne Mathematik

Wir stellen zunächst einmal eine einfachere Frage: Wie haben denn die Ingenieure in der Praxis früher ihre Chips verdrahtet? Gängig ist ein zweistufiges Verfahren.

Im *Global Routing* wird zunächst ein sehr viel gröberer Gittergraph betrachtet; zum Beispiel werden 100 mal 100 Knoten zu einem zusammengefasst. Zwischen zwei benachbarten Knoten kann dann natürlich nicht nur ein Draht, sondern können bis zu hundert parallele Drähte verlaufen. Man packt also jetzt Steinerbäume in ein Gitter mit Kantenkapazitäten. Das ist schon etwas leichter (aber immer noch *NP*-schwer), und die Graphen sind wesentlich kleiner (haben aber immer noch rund 10^7 Knoten). Das Ergebnis des Global Routings wird dann verwendet, um den Suchraum im nachfolgenden *Detailed Routing* einzuschränken. Das bedeutet, für jedes Netz wird nur noch ein relativ kleiner Subgraph des Gitters in Betracht gezogen, in dem der Steinerbaum für dieses Netz gesucht wird.

Abbildung 5 zeigt beispielsweise den vom Global Routing berechneten Korridor für ein Netz (hellgrün) und den vom Detailed Routing berechneten Steinerbaum in diesem Korridor.

Sowohl im Global als auch im Detailed Routing müssen natürlich nach wie vor (viele) Steinerbäume gefunden werden. Selbst das Problem, einen kürzesten Steinerbaum zu finden, ist aber *NP*-schwer. Deshalb werden in der Praxis oft Heuristiken benutzt, zum Beispiel werden sukzessive kürzeste Wege gesucht, die zwei Komponenten eines Netzes miteinander verbinden, die noch nicht verbunden sind. Die Netze werden typischerweise in einer heuristisch gewählten Reihenfolge eins nach dem anderen betrachtet. Wenn es nicht mehr weitergeht, werden Drähte auch wieder entfernt. Dieses Prozedere – ein heuristisches Backtracking – ist als *rip-up and re-route* bekannt.

Wenn die Instanzen nicht zu schwierig sind, kommt man mit solchen recht einfachen Methoden zu einer weitgehend zulässigen Verdrahtung. Die verbleibenden Probleme werden dann manuell repariert. Über die Güte der erhaltenen Lösung kann man so natürlich allenfalls im Nachhinein etwas aussagen. Die eigentlich interessanten Zielfunktionen wurden nicht einmal betrachtet. Oft scheitert das Verfahren komplett, d. h. man findet so überhaupt keine zulässige Verdrahtung. Und auch sonst kann die Rechenzeit viele Tage betragen.

4 Verdrahtung mit Mathematik

Wenn man dieses in der Praxis etablierte Verfahren verbessern will, muss man sich zunächst klarmachen, dass angesichts der enormen Datenmengen nur sehr schnelle Algorithmen in Frage kommen – das heißt aber nicht, dass sie sehr einfach sein müssen. An der grundsätzlichen Aufteilung in Global und Detailed Routing haben wir nichts geändert. Wir haben uns vor allem zwei Fragen gestellt: Kann man im Global Routing eine annähernd optimale Lösung – auch im Hinblick auf die relevanten Zielfunktionen und Nebenbedingungen – schnell genug berechnen? Und kann man im Detailed Routing wenigstens ein einzelnes Netz – innerhalb seines vom Global Routing vorgegebenen Korridors und unter Vermeidung von Konflikten mit früher bearbeiteten Netzen – optimal und schnell genug verdrahten? Beide Fragen können wir heute weitestgehend bejahen, und darauf möchte ich in den beiden folgenden Abschnitten näher eingehen.

4.1 Global Routing mittels Resource Sharing

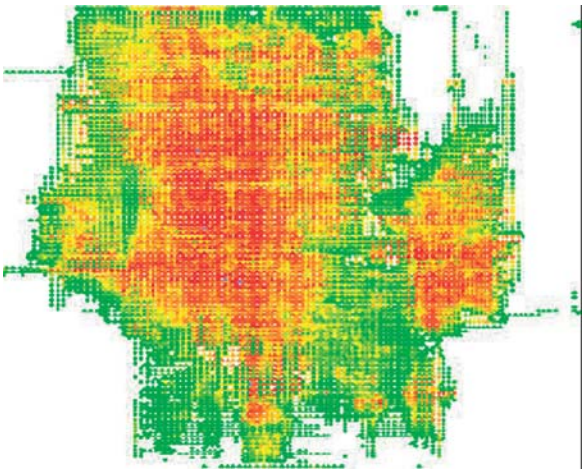
In der Numerik werden schwierige Probleme oft diskretisiert. In der Diskreten Mathematik geht man manchmal den umgekehrten Weg und relaxiert diskrete Probleme, indem man zum Beispiel auch Konvexkombinationen zulässiger Lösungen zulässt. Dadurch entsteht oft ein leichteres Problem. Im Spezialfall des klassischen Global-Routing-Problems (ohne weitere Nebenbedingungen), in dem jedes Netz nur zwei Pins enthält, gelangt man so zum bekannten Multicommodity-Flow-Problem. Raghavan [10] hat gezeigt, dass man aus einer Lösung des Multicommodity-Flow-Problems unter bestimmten Voraussetzungen durch zufälliges Runden eine fast zulässige (und fast optimale) Lösung des (diskreten) Ursprungsproblems erhalten kann.

Das Multicommodity-Flow-Problem kann zwar als Lineares Programm (LP) formuliert und somit in polynomieller Zeit gelöst werden, das hilft jedoch für so große Instanzen nicht weiter. Beginnend mit Shahrokhi und Matula [11] sind aber in den letzten zwanzig Jahren immer bessere sogenannte voll polynomielle Approximationsschemata entwickelt worden, d. h. Algorithmen, die Lösungen finden, die beliebig nah am Optimum liegen (Faktor $1 + \varepsilon$ für beliebiges $\varepsilon > 0$), und deren Laufzeit nur polynomiell von $\frac{1}{\varepsilon}$ abhängt. Im Gegensatz zu allgemeinen LP-Algorithmen sind diese Algorithmen kombinatorischer Natur und auch in der Praxis äußerst schnell.

Einen Algorithmus derselben Art konnten wir nun für ein sehr viel allgemeineres Problem, das so genannte Min-Max-Resource-Sharing-Problem, entwerfen [8]. Dieses Problem geht ganz allgemein davon aus, dass eine endliche Menge von Nutzern um eine endliche Menge beschränkter Ressourcen konkurriert. Für jeden Nutzer gibt es verschiedene Lösungen, die Menge der Lösungen eines Nutzers sei konvex, und jeder Lösung ist über eine konvexe Funktion eine bestimmte Menge jeder Ressource zugeordnet. Es wird vorausgesetzt, dass man zu gegebenen Ressourcenpreisen eine billigste Lösung für einen Nutzer effizient finden kann.

Es zeigt sich nun, dass dieses von Grigoriadis und Khachiyan [3] erstmals studierte Problem wie geschaffen zur Modellierung des Global-Routing-Problems ist. Die Nutzer sind einfach die Netze, und Ressourcen sind die Kanten des Global-Routing-Gitters, aber auch Zeit, Stromverbrauch, und erwartete Fehler in der Fertigung. Unser neuer Algorithmus [8] berechnet in höchstens wenigen Stunden eine fast optimale Lösung (mit garantierter Fehlerschranke) unter Berücksichtigung aller Nebenbedingungen. Das war vorher schlichtweg nicht möglich. Erste Experimente zeigen, dass man damit beispielsweise den Ausschuss in der Fertigung um mehrere Prozentpunkte reduzieren kann, was eine erhebliche Kosteneinsparung bringt.

Abbildung 6 zeigt die Auslastung der Kanten des Global-Routing-Gitters in einer fast optimalen Lösung: rote Kanten sind voll ausgelastet, gelbe etwas weniger, grüne nur etwa zur Hälfte und weiße fast gar nicht. Die anderen Ressourcen sind in diesem Bild nicht dargestellt.



Beispiel für das Ergebnis des Resource-Sharing-Algorithmus. Das Bild zeigt die Ausnutzung der Kantenressourcen. Rote Kanten sind voll ausgelastet, gelbe nicht ganz, grüne weniger, weiße kaum.

4.2 Detailed Routing und kürzeste Wege

Die Kernaufgabe im Detailed Routing, die heute bis zu hundert Millionen mal ausgeführt werden muss, besteht darin, zwei Komponenten eines Netzes durch einen möglichst kurzen Weg miteinander zu verbinden, der aber den bereits zuvor verlegten Drähten und Vias nicht zu nahe kommen darf. Modelliert man den – durch das Global Routing bereits erheblich eingeschränkten – Verdrahtungsbereich als Graphen, so ist dies das bekannte Kürzeste-Wege-Problem, für das Dijkstras [2] Algorithmus bis heute das Standardverfahren ist. Mit geeigneten Datenstrukturen findet er einen kürzesten Weg in $O(n \log n)$ Rechenschritten, wobei n die Anzahl der Knoten des Graphen ist (dessen maximaler Grad hier sechs ist).

Dies sieht zwar auf den ersten Blick sehr schnell aus, und reicht für die meisten Anwendungen auch völlig aus, ist hier aber viel zu langsam. Eine bekannte Technik (die beispielsweise auch in Navigationssystemen eingesetzt wird) besteht in der so genannten zielorientierten Suche. Diese verringert die theoretische Laufzeitschranke zwar nicht, kann aber in der Praxis eine große Beschleunigung bringen, wenn gute Entfernungsabschätzungen vorliegen.

Um akzeptable Rechenzeiten zu erhalten, ist es aber im Detailed Routing zusätzlich notwendig, die Struktur des Graphen auszunutzen. Erstens haben wir einen dreidimensionalen Gittergraphen mit relativ wenigen Ebenen (zur Zeit ungefähr zehn). Zweitens wird der Verdrahtungsbereich durch relativ wenige Rechtecke beschrieben; diese entsprechen dem Ergebnis des Global Routing. Drittens geht der Graph aus einem in diesem Verdrahtungsbereich weitgehend vollständigen Gitter dadurch

hervor, dass Knoten mit Konflikten zu früher verlegten Drähten entfernt werden.

Legt man die Drähte auf einer Ebene nun fast ausschließlich parallel zueinander (was ohnehin für eine dichte Packung zweckmäßig ist), so kann der Graph durch vergleichsweise wenige Intervalle aufeinanderfolgender Punkte gleicher Eigenschaft beschrieben werden.

Man kann den Graphen also durch drei Hierarchieebenen beschreiben: oben die Rechtecke, die den Global-Routing-Korridor beschreiben, dann die Intervalle, und zuunterst die einzelnen Knoten. In [9] haben wir gezeigt, wie man die zielorientierte Variante von Dijkstras Algorithmus auf derartigen Graphen so implementieren kann, dass man die unterste Hierarchieebene nie betrachten muss und die Laufzeit deshalb nur von der Zahl der Intervalle abhängt – diese ist hier rund 50 mal geringer als die Zahl der Knoten.

Dank dieses neuen Algorithmus muss man nicht darauf verzichten, garantiert kürzeste Wege zu finden. Unser Verdrahtungsprogramm BonnRoute ist sogar schneller als kommerzielle Programme, obwohl diese für die Kernaufgabe Heuristiken verwenden und deshalb oft nicht den kürzesten Weg finden, worunter natürlich die Qualität der Lösung leidet.

5 Ausblick

An interessanten offenen mathematischen Problemen mangelt es nach wie vor nicht. Einige haben wir in [5] zusammengetragen, und nahezu wöchentlich begegnen wir neuen. Neue Probleme entstehen zum einen durch die weiterhin rasante technologische Entwicklung, aber auch durch neue algorithmische Ansätze, die bisher nicht studierte Teilprobleme aufwerfen.

Andererseits gibt es auch Probleme, die irrelevant werden, weil sie von der technologischen Entwicklung überholt wurden. Beispielsweise wurden in den 1980er Jahren Verdrahtungsprobleme mit zwei Verdrahtungsebenen intensiv studiert. Das war damals Stand der Technik, ist aber heute hinfällig.

Das Gebiet ist also ständig in Bewegung. Daher ist der möglichst direkte Kontakt zu den Designingenieuren so wichtig. Ich glaube, hier kann man mit Fug und Recht behaupten, dass Mathematik und Anwendung sich gegenseitig befruchten.

Dank

Chip-Design ist Teamwork. Daher gilt mein Dank allen Kollegen, Mitarbeitern und Studenten in unserem Bonner Institut, allen voran Bernhard Korte. Außerdem danke ich Ina Prinz, ohne die unsere Bilder nur halb so schön wären.

Anmerkungen

1. Moore [7] hatte zunächst, bereits 1965, eine jährliche Verdoppelung bis 1975 prognostiziert, und diese Prognose dann angepasst.
2. Die Technologiegenerationen werden mit der Breite der kleinsten Strukturen bezeichnet, die gefertigt werden können. Aktuelle Höchstleistungschips werden in 65-nm- und 45-nm-Technologie gefertigt, die nächsten Generationen (32 nm und 22 nm) sind in Vorbereitung. Von einer Generation zur nächsten sinken alle Längenmaße ungefähr um den Faktor $\sqrt{2}$, was theoretisch eine Verdoppelung der Anzahl der Bauteile pro Quadratzentimeter ermöglicht. Die Größe der Chips wächst kaum; sie liegt meist bei wenigen Quadratzentimetern.
3. Es hat zwar auch ernsthafte Versuche gegeben, zusätzlich diagonale Drähte zu ermöglichen, sie sind aber bis dato alle gescheitert. Der immense Aufwand scheint sich nicht zu rentieren.
4. Natürlich setzt dies voraus, dass die Pins so angeschlossen werden können, was nicht immer der Fall ist. Die damit verbundenen Probleme wollen wir hier der Einfachheit halber ignorieren; deren Lösung ist lokaler Natur und stört das Gesamtbild nicht.

Literatur

[1] C. J. Alpert, D. P. Mehta, S. S. Sapatnekar (Hg.): Handbook of Algorithms for VLSI Physical Design Automation. CRC Press 2009

[2] E. W. Dijkstra: A note on two problems in connexion with graphs. Numerische Mathematik 1 (1959), 269–271

[3] M. D. Grigoriadis, L. D. Khachiyan: Fast approximation schemes for convex programs with many blocks and coupling constraints. SIAM Journal on Optimization 4 (1994), 86–107

[4] S. Held, B. Korte, D. Rautenbach, J. Vygen: Combinatorial optimization in VLSI design. Erscheint in: "Combinatorial Optimization: Methods and Applications" (V. Chvátal, N. Sbihi, Hg.), IOS Press

[5] B. Korte, J. Vygen: Combinatorial problems in chip design. In: Building Bridges Between Mathematics and Computer Science (M. Grötschel, G.O.H. Katona, Hg.), Springer, Berlin 2008, pp. 333–368

[6] B. Korte, D. Rautenbach, J. Vygen: BonnTools: Mathematical innovation for layout and timing closure of systems on a chip. Proceedings of the IEEE 95 (2007), 555-572

[7] G. E. Moore: Cramming more components onto integrated circuits. Electronics 38 (1965), 114–117

[8] D. Müller, J. Vygen: Faster min-max resource sharing and applications. Technical Report No. 08987, Research Institute for Discrete Mathematics, University of Bonn, 2008

[9] S. Peyer, D. Rautenbach, J. Vygen: A generalization of Dijkstra's shortest path algorithm with applications to VLSI routing. Erscheint in Journal of Discrete Algorithms

[10] P. Raghavan: Probabilistic construction of deterministic algorithms: approximating packing integer programs. Journal of Computer and System Sciences 37 (1988), 130–143

[11] F. Shahrokhi, D. W. Matula: The maximum concurrent flow problem. Journal of the ACM 37 (1990), 318–334

Professor Dr. Jens Vygen, Forschungsinstitut für Diskrete Mathematik, Universität Bonn, Lennéstraße 2, 53113 Bonn.
vygen@or.uni-bonn.de