Optimizing Yield in Global Routing

Dirk Müller
Research Institute for Discrete Mathematics
University of Bonn
Lennéstr. 2, 53113 Bonn, Germany
mueller@or.uni-bonn.de

ABSTRACT

We present the first efficient approach to global routing that takes spacing-dependent costs into account and provably finds a near-optimum solution including these costs. We show that this algorithm can be used to optimize manufacturing yield. The core routine is a parallelized fully polynomial approximation scheme, scaling very well with the number of processors. We present results showing that our algorithm reduces the expected number of defects in wiring by more than 10 percent on state-of-the-art industrial chips.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids—*Placement and Routing*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph Algorithms, Network Problems*

General Terms

Algorithms

Keywords

Multi-commodity flows, Steiner tree packing, VLSI routing, yield optimization

1. INTRODUCTION

Because of the huge size of VLSI routing instances, a global routing step is usually performed before detailed routing, defining an area for each net to which the search space will be restricted. In its simplest form, the global routing problem amounts to packing Steiner trees in capacitated undirected graphs. Traditionally, the main objective was netlength minimization. However, with the decrease of feature sizes, effects depending on wire spacing become more and more important: First, manufacturing yield improves with a better spreading of wires, and second, coupling capacitance becomes increasingly important for power consumption and signal propagation delay. On the other hand, spacing usually can be increased only at the expense of netlength, which also has a negative effect on yield, power and timing. Recent papers on global routing (Xu et al. [13], Jing et al. [7], Ho et al. [5]) propose heuristics partially adressing this tradeoff, but there is no approach yet that gives any performance guarantee (see [6] for a survey).

Deciding if a global routing instance has a feasible solution or not is NP-complete even when restricted to two-terminal nets which have to be routed in planar grid graphs with unit capacities [9]. Raghavan and Thompson [10] proposed to solve an LP relaxation first and then apply randomized rounding to obtain an integral solution whose maximum violation of the packing constraints can be bounded. However, given today's huge instance sizes, even solving the LP relaxation exactly is far too slow in practice. Therefore approximation algorithms are interesting. Shahrokhi and Matula [11] proposed the first fully polynomial approximation scheme for multicommodity flows, which then was applied to global routing by Carden, Li and Cheng [2]. Garg and Könemann [4] presented a simpler and more efficient approximation scheme, modified and applied to global routing by Albrecht [1].

Recently, Vygen [12] proposed the first global routing algorithm taking spacing-dependent costs into account, guaranteeing to find a solution with almost optimum total cost. However, this is a purely theoretical work that does not consider manufacturing yield, and the algorithm cannot be parallelized efficiently. In this paper, we describe an enhanced and parallelized version of the algorithm that efficiently optimizes manufacturing yield and scales very well with the number of processors. We also present results on a number of state-of-the-art VLSI designs from industry, showing that the expected number of defects in wiring can be reduced by more than 10 percent on average.

This work is organized as follows. In section 2, a formal description of the global routing problem with spacing-dependent costs is given. Section 3 presents a parallelized fully polynomial approximation scheme for efficiently solving a fractional linear programming relaxation of this problem. Section 4 briefly describes the randomized rounding step applied to obtain an integral solution. Section 5 introduces the defect model that we assume for measuring defect sensitivities and shows how yield can be optimized with our algorithm. In section 6, we present results on several current industry designs which demonstrate the yield optimization capabilities of our algorithm and show that indeed our algorithm is parallelized very efficiently. We conclude the paper with some discussion and outlook on future work.

ICCAD'06, November 5-9, 2006, San Jose, CA

2. THE GLOBAL ROUTING PROBLEM

This work neither assumes a routing grid, nor is it limited to Manhattan routing, so also diagonal wiring is possible. We construct a global routing graph by partitioning the chip area into regions (forming the vertices of the graph) and joining adjacent regions by an edge. Each edge is assigned a capacity value that indicates how many wires of unit width can join the two regions such that all design rules are met.

In the following, let G be the global routing graph, with edge capacities $u : E(G) \to \mathbb{R}_+$. Let $\mathcal{N}$ be the set of nets to be routed. For each pair $(e, N) \in E(G) \times \mathcal{N}$ we are given a constant $w_{e,N}$ which denotes the width of a wire of net N using edge e. This allows for choosing wire types not only depending on the net, but also plane- or even region-dependent. We also have a cost interval $[c^{\min}_{e,N}, c^{\max}_{e,N}]$ for each $(e, N) \in E(G) \times \mathcal{N}$, assuming that the maximum cost $c^{\max}_{e,N}$ is attained at minimum spacing to neighbouring wires and reduces linearly if extra spacing is assigned, until a specified amount of spacing $s_{e,N}$ corresponding to costs $c^{\min}_{e,N}$ is reached.

Now, for each net $N \in \mathcal{N}$ we construct a global routing net as follows. First, for each pin we determine the set of vertices of G corresponding to regions intersected by its shapes. Then, iteratively, we merge each pair of sets with nonempty intersection until no such pair exists any more. Each of the resulting sets is called a global routing pin of N.

The task now is to find a connected subgraph Y_N of G for each $N \in \mathcal{N}$, intersecting all of its global routing pins, and extra space assignments such that edge capacities are respected and total cost is minimized. It is possible to restrict the set $\mathcal{Y}_N$ of feasible subgraphs Y_N of G for net N, e. g. to Steiner trees or, for timing critical nets, to Elmore-delay-optimal Steiner trees. However, this is not important for our approach. Note that $\mathcal{Y}_N$ does not need to be specified explicitely, but can be given implicitely by an oracle function returning a cost optimal element from this set.

Additionally, cost bounds can be assigned to subsets of $\mathcal{N}$, so we are given a family $\mathcal{M}$ of subsets of $\mathcal{N}$ with bounds $U : \mathcal{M} \to \mathbb{R}_+$ and weights $\sigma(M, N) \in \mathbb{R}_+$ for $N \in M \in \mathcal{M}$. We require that $\mathcal{N} \in \mathcal{M}$.

Let $\Delta_{e,N} := c^{\max}_{e,N} - c^{\min}_{e,N}$. We can formulate the global routing problem more formally now: the task is to find an $Y_N \in \mathcal{Y}_N$ and numbers $0 \le y_{e,N} \le 1$ for each $N \in \mathcal{N}$ and $e \in E(Y_N)$ (denoting the fraction of possible extra space $s_{e,N}$ to be assigned to net N on edge e), such that

$$\sum_{N \in \mathcal{N}: e \in E(Y_N)} (w_{e,N} + y_{e,N} s_{e,N}) \le u(e) \tag{1}$$

for each edge $e \in E(G)$,

$$\sum_{N \in M} \sigma(M, N) \sum_{e \in E(Y_N)} (c^{\max}_{e,N} - y_{e,N} \Delta_{e,N}) \le U(M) \tag{2}$$

for $M \in \mathcal{M}$, and such that

$$\sum_{N \in \mathcal{N}} \sigma(\mathcal{N}, N) \sum_{e \in E(Y_N)} (c^{\max}_{e,N} - y_{e,N} \Delta_{e,N}) \tag{3}$$

is minimum.

3. FRACTIONAL GLOBAL ROUTING

Since it is hard to optimize the integral version of the global routing problem, we start with a fractional linear programming relaxation.

3.1 LP Formulation

We first observe that it is sufficient to find a feasible solution as we can impose an arbitrary upper bound $U(\mathcal{N})$ on (3) and perform binary search to find the optimum value of $U(\mathcal{N})$, in each step solving a linear program that maximizes the minimum slack in (1) and (2), followed by randomized rounding (cf. section 4).

Our experiments show that binary search is not needed in practice since good lower bounds on (3) can be computed and the algorithm comes close to these bounds within a few percent on all of our test cases, so it suffices to solve one linear program. In either case it is worthwile to change the objective from minimizing (3) to maximizing the minimum slack in (1) and (2) because this can considerably improve running time of detailed routing and in many cases increasing slacks in the cost bounding inequalities is desirable.

So we approximately solve the following linear program:

$$\begin{aligned}
&\min \lambda \quad \text{subject to} \\
&\sum_{Y \in \mathcal{Y}_N} x_{N,Y} = 1 && \text{for } N \in \mathcal{N} \\
&\sum_{N \in M} \sigma(M,N) \left(\sum_{Y \in \mathcal{Y}_N} \sum_{e \in E(Y)} x_{N,Y} c^{\max}_{e,N} - \sum_{e \in E(G)} y_{e,N} \Delta_{e,N} \right) \le \lambda U(M) && \text{for } M \in \mathcal{M} \quad (4) \\
&\sum_{N \in \mathcal{N}} \left(\sum_{\substack{Y \in \mathcal{Y}_N: \\ e \in E(Y)}} x_{N,Y} w_{e,N} + y_{e,N} s_{e,N} \right) \le \lambda u(e) && \text{for } e \in E(G) \\
&y_{e,N} \le \sum_{\substack{Y \in \mathcal{Y}_N: \\ e \in E(Y)}} x_{N,Y} && \text{for } e \in E(G), N \in \mathcal{N} \\
&y_{e,N} \ge 0 && \text{for } e \in E(G), N \in \mathcal{N} \\
&x_{N,Y} \ge 0 && \text{for } N \in \mathcal{N}, Y \in \mathcal{Y}_N
\end{aligned}$$

The corresponding dual LP is:

$$\begin{aligned}
&\max \sum_{N \in \mathcal{N}} z_N \quad \text{subject to} \\
&\sum_{e \in E(G)} u(e) \omega_e + \sum_{M \in \mathcal{M}} U(M) \mu_M = 1 \\
&z_N \le \sum_{e \in E(Y)} \left(c^{\max}_{e,N} \sum_{\substack{M \in \mathcal{M}: \\ N \in M}} \sigma(M,N) \mu_M + w_{e,N} \omega_e - \chi_{e,N} \right) && \text{for } N \in \mathcal{N}, Y \in \mathcal{Y}_N \\
&\chi_{e,N} \ge \Delta_{e,N} \sum_{\substack{M \in \mathcal{M}: \\ N \in M}} \sigma(M,N) \mu_M - s_{e,N} \omega_e && (5) \\
& && \text{for } e \in E(G), N \in \mathcal{N} \\
&\chi_{e,N} \ge 0 && \text{for } e \in E(G), N \in \mathcal{N} \\
&\omega_e \ge 0 && \text{for } e \in E(G) \\
&\mu_M \ge 0 && \text{for } M \in \mathcal{M}
\end{aligned}$$

The dual LP enables us to compute a lower bound on the optimum LP value:

LEMMA 1. *Let $\omega_e \in \mathbb{R}_+$ for $e \in E(G)$ and $\mu_M \in \mathbb{R}_+$ for $M \in \mathcal{M}$, and let us define* edge costs

$$\psi_{e,N} := \min_{\delta \in \{0,1\}} \Big(\big(c_{e,N}^{\max} - \delta\Delta_{e,N}\big)\gamma_N + \big(w_{e,N} + \delta s_{e,N}\big)\omega_e \Big), \tag{6}$$

where $\gamma_N := \sum_{M \in \mathcal{M}: N \in M} \sigma(M,N)\mu_M$. Then

$$\frac{\sum_{N \in \mathcal{N}} \min_{Y \in \mathcal{Y}_N} \sum_{e \in E(Y)} \psi_{e,N}}{\sum_{e \in E(G)} u(e)\omega_e + \sum_{M \in \mathcal{M}} U(M)\mu_M}$$

is a lower bound on the optimum LP value.

PROOF. Set $\chi_{e,N} := \max\{0, \Delta_{e,N}\gamma_N - s_{e,N}\omega_e\}$, and

$$\begin{aligned} z_N &:= \min_{Y \in \mathcal{Y}_N} \sum_{e \in E(Y)} \psi_{e,N} \\ &= \min_{Y \in \mathcal{Y}_N} \sum_{e \in E(Y)} \Big(c_{e,N}^{\max}\gamma_N + w_{e,N}\omega_e - \chi_{e,N} \Big). \end{aligned}$$

Then we get a feasible solution (z, μ, ω, χ) of (5) after dividing all variables by $\sum_{e \in E(G)} u(e)\omega_e + \sum_{M \in \mathcal{M}} U(M)\mu_M$. □

3.2 Parallel Fractional Global Routing Algorithm

Now we are ready to present the approximation scheme for approximately solving (4) and (5). It is a primal-dual algorithm that takes four parameters $0 < \epsilon, \epsilon_1, \epsilon_2 < 1$ and $t \in \mathbb{N}$, which control the approximation guarantee and running time of the algorithm, and a parameter $\Pi \in \mathbb{N}$ specifying the number of processors to be used in parallel. Furthermore, we assume to have a lower bound $\mathcal{L}_N$ on $\sum_{e \in E(Y)} c_{e,N}^{\min}$ of any Steiner tree $Y \in \mathcal{Y}_N (N \in \mathcal{N})$. The quality of this lower bound is important for obtaining good running times in practice. Often a nontrivial lower bound can be computed in constant time using distances between the terminals.

We have implemented a shared memory multithreaded version of the algorithm, but also implementations using processes distributed on different machines should be quite efficient because the need for communication between processes is very low.

Parallel Fractional Global Routing Algorithm
Input: An instance of the *Global Routing Problem*, $t \in \mathbb{N}$, $\epsilon, \epsilon_1, \epsilon_2 \in \mathbb{R}_+$, and $\Pi \in \mathbb{N}$.
Output: A feasible solution to (4).

① Set $\alpha_e := 0$ and $\omega_e = \frac{1}{u(e)}$ for $e \in E(G)$.
Set $\beta_M := 0$ and $\mu_M = \frac{1}{U(M)}$ for $M \in \mathcal{M}$.
Set $x_{N,Y} := 0$ for $N \in \mathcal{N}$ and $Y \in \mathcal{Y}_N$.
Set $y_{e,N} := 0$ for $e \in E(G)$ and $N \in \mathcal{N}$.
Set $Y_N := \emptyset$ for $N \in \mathcal{N}$.
ResetVariableUpdates.
② **For** $p := 1$ to t **do**:
Create a partition $\mathcal{N}_1 \dot\cup \ldots \dot\cup \mathcal{N}_\Pi = \mathcal{N}$.
For $\pi \in \{1, \ldots, \Pi\}$ **do in parallel**:
RouteNets(π).
CollectVariableUpdates.
ResetVariableUpdates.
③ Set $x_{N,Y} := \frac{1}{t} x_{N,Y}$ for $N \in \mathcal{N}$ and $Y \in \mathcal{Y}_N$.
Set $y_{e,N} := \frac{1}{t} y_{e,N}$ for $e \in E(G)$ and $N \in \mathcal{N}$.

Procedure RouteNets(π):
For $N \in \mathcal{N}_\pi$ **do**:
Let $\psi_{e,N}$ be defined as in (6).
If $Y_N = \emptyset$ or $\sum_{e \in E(Y_N)} \psi_{e,N} > (1+\epsilon_1) z_N$ **then**:
Let $Y_N \in \mathcal{Y}_N$ with $\sum_{e \in E(Y_N)} \psi_{e,N} \le (1+\epsilon_2) \min_{Y \in \mathcal{Y}_N} \sum_{e \in E(Y)} \psi_{e,N}$.
Set $z_N := \sum_{e \in E(Y_N)} \psi_{e,N}$.
UpdateVariables(N, π).

Procedure UpdateVariables(N, π):
Set $x_{N,Y_N} := x_{N,Y_N} + 1$.
For $e \in E(Y_N)$ **do**:
If $\Delta_{e,N}\gamma_N < s_{e,N}\omega_e$ **then** $\delta := 0$ **else** $\delta := 1$.
Set $y_{e,N} := y_{e,N} + \delta$.
Set $\alpha_e^\pi := \alpha_e^\pi + w_{e,N} + \delta s_{e,N}$.
For $M \in \mathcal{M}$ with $N \in M$ **do**:
Set $\beta_M^\pi := \beta_M^\pi + \sigma(M,N)\big(c_{e,N}^{\max} - \delta\Delta_{e,N}\big)$.

Procedure CollectVariableUpdates:
For each $e \in E(G)$ **do**:
For $\pi := 1$ to Π **do**:
Set $\alpha_e := \alpha_e + \alpha_e^\pi$.
Set $\omega_e := \frac{1}{u(e)} e^{\epsilon \frac{\alpha_e}{u(e)}}$.
For each $M \in \mathcal{M}$ **do**:
For $\pi := 1$ to Π **do**:
Set $\beta_M := \beta_M + \beta_M^\pi$.
Set $\mu_M^{\text{old}} := \mu_M$.
Set $\mu_M := \frac{1}{U(M)} e^{\epsilon \frac{\beta_M}{U(M)}}$.
For each $N \in \mathcal{N}$ **do**:
For each $M \in \mathcal{M}$ with $N \in M$ **do**:
$z_N := z_N + (1+\epsilon_2)\mathcal{L}_N \sigma(M,N)\big(\mu_M - \mu_M^{\text{old}}\big)$.

Procedure ResetVariableUpdates:
For $\pi := 1$ to Π **do**:
For each $e \in E(G)$ **do**:
Set $\alpha_e^\pi := 0$.
For each $M \in \mathcal{M}$ **do**:
Set $\beta_M^\pi := 0$.

Let us call one iteration of the outer loop (step ② of the algorithm) a *phase*. While the previous algorithms ([1, 12]) work sequentially, routing the nets in a certain predetermined order and updating the dual variables after each completion of a net, our algorithm does not need an ordering of the nets. It turns out that for the analysis of the algorithm it is not important if updating the μ, ω and z variables is done immediately or delayed until the end of the current phase. Delaying updates drastically reduces interdependence between threads since changes of variables that are accessed by more than one thread occur only at the end of a phase. This makes it possible to implement the algorithm using almost no locking and scaling very well with the number of processors used.

Besides that, aggregating and delaying the updates of the z variables until the end of the current phase eliminates the innermost loop of the algorithm in [12] which easily dominated running time. Therefore not only parallelization becomes more efficient, but also total CPU time is reduced.

The following theorem shows that the presented algorithm is a fully polynomial approximation scheme for solving the fractional global routing problem:

THEOREM 2. *Given an* approximation parameter ϵ_0*, one can find* $\epsilon, \epsilon_1, \epsilon_2 \in \mathbb{R}_+$ *and* $t \in O\left(\frac{\log(|E(G)|+|\mathcal{M}|)}{\epsilon_0^2}\right)$ *such that with parameters* $\epsilon, \epsilon_1, \epsilon_2$ *and* t *the presented algorithm finds a* $(1+\epsilon_0)$*-optimal feasible solution to (4).*

PROOF. Our algorithm makes use of the α and β variables to record each thread's contribution to dual variable updates. At the end of a phase, the contributions from each thread are summed up in order to compute new values for the ω, μ and z dual variables. With some modifications, the analysis of the algorithm in [12] can be made to work also for our algorithm with delayed updates of dual variables: Clearly, estimating the increase of the μ and ω variables in inequalities (9) and (10) of [12] does not need to be done net by net. If all nets are considered at once, nothing changes except that the bound on the increase has to be expressed in terms of dual variable values at the beginning of the current phase. This, however, does not hurt in the continuation of the proof. □

Table 1 shows that our parallel algorithm scales very well with the number of processors: We did experiments with two large 130 nm chips (with 2.5 million and 2.8 million nets, respectively) on an IBM S85 machine with 96 GB memory and 18 CPU's running at 600 MHz. This machine is slower than a current Opteron machine by a factor of 3.5 to 4, so much better running times can be obtained on current machines. We routed the chips using 1, 4, 8 and 16 threads in parallel. The corresponding speedups in some cases are even slightly higher than the number of processors. This is probably due to caching effects.

3.3 Future cost

Finding near-optimum Steiner trees in the *RouteNets* procedure is the core routine in our algorithm that consumes most running time. Since all edge weights are nonnegative, we can apply the algorithm of Dijkstra for two-terminal nets. For nets with more than two terminals, we first find geometrically optimum Steiner trees and then perform a series of path searches connecting the terminals and Steiner points to each other, also applying Dijkstra's algorithm. Alternatively, we can use the algorithm of Dreyfus and Wagner [3] to find optimum Steiner trees directly (note that this algorithm in fact is a generalization of bidirectional search).

In either case, an important factor speeding up the algorithm is the use of a future cost estimate, which is a lower bound of the distance of vertices to a given set of target vertices. Suppose we look for a path from s to t in an undirected graph G with vertex set V, edge set E and edge lengths $c : E \to \mathbb{R}_+$. Let $l(v)$ be a lower bound on the distance from v to t for any $v \in V$, satisfying the natural condition $l(v) \le c(e)+l(w)$ for each edge $e = \{v, w\} \in E$. If we create G' from G by replacing each edge with two oppositely directed edges connecting the same vertices and define $c'((v, w)) := c(\{v, w\})-l(v)+l(w)$ for each edge (v, w) in G', then a shortest path from s to t in (G, c) is also a shortest path in (G', c') and vice versa. If l is close to the exact distance, finding a shortest path in (G', c') can be done much faster than in (G, c). With exact future cost estimates, only vertices actually lying on a shortest path are labelled (the cost of a shortest path in (G', c') is zero then).

For yield optimization, the $c^{\min}$ values are proportional to edge lengths (cf. section 5), therefore we can compute future cost based on L_1-distances.

Chip	# Threads	Runtime (hh:mm)	Speedup
Ulrich	1	24:31	
(2.8M nets,	4	05:35	4.39
130 nm)	8	02:49	8.70
	16	01:28	16.72
Hermann	1	07:30	
(2.5M nets,	4	01:52	4.02
130 nm)	8	00:57	7.89
	16	00:30	15.00

Table 1: Parallelization Speedup (on an IBM S85 at 600 MHz)

3.4 Balancing geometrical and congestion parts of edge costs

The edge costs $\psi_{e,N}$ for net N using edge e defined in (6) consist of a term $\left(c_{e,N}^{\max} - \delta\Delta_{e,N}\right)\gamma_N$ $(\delta \in \{0, 1\})$ which for reasonable cost measures is approximately proportional to the geometrical length of the edge, and a second part $\left(w_{e,N} + \delta s_{e,N}\right)\omega_e$ that captures congestion costs. Of course, future cost can consider only the first part (with $\delta = 1$).

Typically (depending on the size of the regions that form the vertices of the global routing graph) edge capacities are not much larger than 100 to 200 units. On the other hand, often a net N has large $\min_{M:N\in M\in\mathcal{M}} |M|$. Therefore, since $\omega_e := \frac{1}{u(e)}(e \in E(G))$ and $\mu_M := \frac{1}{U(M)}(M \in \mathcal{M})$ after initialization, edge costs are dominated by the congestion part in the first phases, even in regions with very low capacity utilization. This means that future cost becomes very poor and leads to high running times.

Fortunately, we can balance the geometrical and congestion parts of the edge costs without actually changing the routing instance: It suffices to duplicate the capacitance constraint (2) for the group $\mathcal{N}$ of all nets a certain number of times in the primal LP (4). This does not change the linear program, and is equivalent to a multiplication of the dual variable $\mu_{\mathcal{N}}$. In this way, the geometrical edge cost parts get higher weight without adding complexity to the algorithm, resulting in better future cost and running times. In our implementation, we choose $\mu_{\mathcal{N}}$ such that, taking the average over all edges, geometrical and congestion parts of the edge costs are equal after initialization in step ①.

4. RANDOMIZED ROUNDING

To obtain an integral solution, we use randomized rounding as in [12]. Let (x, y, λ) be a fractional solution to the primal LP. To obtain an integral solution $(\hat{x}, \hat{y}, \hat{\lambda})$, we do the following. First, we choose $Y \in \mathcal{Y}_N$ as Y_N with probability $x_{N,Y}$ (independently for all $N \in \mathcal{N}$). Then we set $\hat{x}_{N,Y_N} := 1$ and $\hat{x}_{N,Y} := 0$ for $Y \in \mathcal{Y}_N \setminus \{Y_N\}$. Now, set $\hat{y}_{e,N} := \frac{y_{e,N}}{\sum_{Y\in\mathcal{Y}_N | e\in E(Y)} x_{N,Y}}$ if $e \in E(Y_N)$, and $\hat{y}_{e,N} := 0$ otherwise. Finally, we choose $\hat{\lambda}$ minimum possible such that $(\hat{x}, \hat{y}, \hat{\lambda})$ is a feasible solution to (4). We have:

THEOREM 3. *Let* $\Lambda \le \frac{U(M)}{\sigma(M,N)\sum_{e\in E(Y)} c_{e,N}^{\max}}$ *for* $N \in M \in \mathcal{M}$ *and* $Y \in \mathcal{Y}_N$*, and* $\Lambda \le \frac{u(e)}{w_{e,N}+s_{e,N}}$ *for* $N \in \mathcal{N}$*. Moreover, suppose that* $|\mathcal{M}| + |E(G)| < e^{\lambda\Lambda}$*. Then* $\hat{\lambda} \le \lambda\left(1 + (e-1)\sqrt{\frac{\ln(|\mathcal{M}|+|E(G)|}{\lambda\Lambda}}\right)$ *with probability at least* $\frac{1}{2}$*.* □

In our experiments, only very few upper bounds are violated after randomized rounding, and only a few ripup and reroute steps are needed to correct them.

5. YIELD OPTIMIZATION

Critical area analysis as in [8] was used to evaluate the yield optimization results of the algorithm presented in this paper. Only the chip-level routing was considered, but not the internal structure of custom macros and library cells.

In this approach, we are given a defect size distribution provided by manufacturing. In our case, the distribution is

$$f(r) := \begin{cases} 0, r < r_0 \\ \frac{c}{r^3}, r \geq r_0 \end{cases}$$

for some $r_0 \in \mathbb{R}_+$ smaller than the smallest possible particle that can cause a fault, and c such that $\int_0^\infty f(r)\mathrm{d}r = 1$.

Then the *critical area* w. r. t. short-defects on plane z is

$$C^z_{\text{short}} := \kappa^z_{\text{short}} \int_x \int_y \int_{t_{\text{short}}(x,y,z)}^{\infty} f(r)\,\mathrm{d}r\,\mathrm{d}y\,\mathrm{d}x, \quad (7)$$

where $t_{\text{short}}(x, y, z)$ is the smallest size of a particle that causes a short-defect at location (x, y, z), and κ^z_{short} is a weighting factor provided by manufacturing that encodes the relative probability of short defects on plane z. Similarly,

$$C^z_{\text{open}} := \kappa^z_{\text{open}} \int_x \int_y \int_{t_{\text{open}}(x,y,z)}^{\infty} f(r)\,\mathrm{d}r\,\mathrm{d}y\,\mathrm{d}x, \quad (8)$$

where $t_{\text{open}}(x, y, z)$ is the smallest size of a particle that causes an open-defect at location (x, y, z), and κ^z_{open} is provided by manufacturing.

For simple situations like a straight wire without neighbours or straight wires running in parallel, these values can be computed analytically. For yield optimization, we can compute spacing-dependent costs as follows: For a wire of width w in plane z surrounded by neighbouring wires at distance d on both sides, we define the contribution of this wire to critical area by integrating (7) and (8) over the Voronoi region of this wire, i. e. the set of points in plane z where this wire is the closest object. Ignoring wire ends and assuming $r_0 < \min\{\frac{w}{2}, \frac{d}{2}\}$, we thus get a contribution of

$$\begin{aligned} \rho(w,d) &:= 2c \int_0^{\frac{d+w}{2}} \int_{x+\frac{w}{2}}^{\infty} \frac{1}{r^3}\,\mathrm{d}r\,\mathrm{d}x \\ &= 2c \int_0^{\frac{d+w}{2}} \frac{2}{(2x+w)^2}\,\mathrm{d}x \\ &= 2c \left(\frac{1}{w} - \frac{1}{d+2w}\right) \end{aligned}$$

to open critical area for a unit-length piece of wire if $\kappa^z_{\text{open}} = 1$. Similarly, for $\kappa^z_{\text{short}} = 1$ the contribution to short critical area is

$$\tau(w,d) := 2c \left(\frac{1}{d} - \frac{1}{w+2d}\right).$$

Fig. 1 shows $\rho(w, d)$, $\tau(w, d)$ and $\rho(w, d) + \tau(w, d)$ for $w = 0.5$ tracks and d from 0.5 to 5 tracks. Apparently, if short critical area is not weighted considerably higher than open critical area, not much can be gained in the sum of both by having more than one track of extra space.

Let l_e be the length of edge e. The maximum cost $c^{\max}_{e,N}$ for a wire of net N using edge e is attained with two neighbours

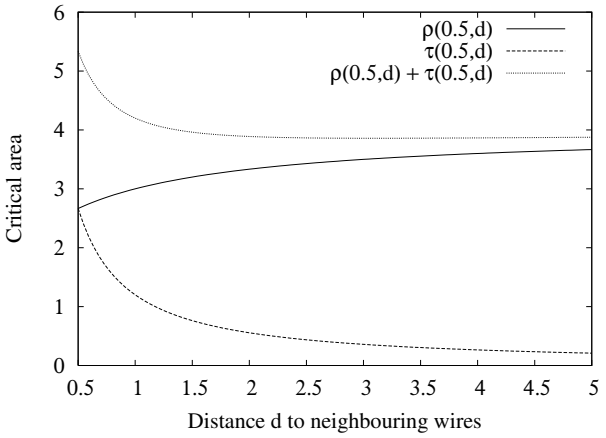

Figure 1: Critical area contribution of a unit-length piece of wire of width 0.5 tracks

running in parallel at minimum distance $d^{\min}_{e,N}$, so we set

$$c^{\max}_{e,N} := l_e \left(\kappa^{z(e)}_{\text{open}} \rho(w_{e,N}, d^{\min}_{e,N}) + \kappa^{z(e)}_{\text{short}} \tau(w_{e,N}, d^{\min}_{e,N})\right)$$

and (with $d^{\max}_{e,N} := d^{\min}_{e,N} + s_{e,N}$)

$$c^{\min}_{e,N} := l_e \left(\kappa^{z(e)}_{\text{open}} \rho(w_{e,N}, d^{\max}_{e,N}) + \kappa^{z(e)}_{\text{short}} \tau(w_{e,N}, d^{\max}_{e,N})\right),$$

where $z(e)$ is the plane in which edge e lies.

Note that our model requires costs to be linear in the amount of extra spacing. This of course is a simplification. Moreover, it may not always be possible to realize the global routing solution in detailed routing such that all wires which have been assigned extra spacing are put next to each other.

To compute the expected number of faults within chip-level wiring of a complete VLSI chip, we use a Monte Carlo dot-throwing approach as in [8] to determine the planewise critical area values, sum up these values and finally multiply the result by chip area.

6. RESULTS

We have run the algorithm presented in this paper on a number of current VLSI designs from industry. Table 2 shows our testbed, consisting of three 90nm designs, six 130nm designs, and one 180nm design. All designs use horizontal and vertical wiring only and have 6–8 routing planes.

In tables 3 and 4, we compare objective values after global routing for different optimization objectives on two of the

Chip	Technology	Image Size (in 1000 channels)	# Nets (in 1000)
Edgar	90 nm	40 x 40	772
Hannelore	90 nm	36 x 33	140
Paul	90 nm	24 x 24	68
Monika	130 nm	35 x 35	1502
Garry	130 nm	26 x 26	827
Heidi	130 nm	23 x 23	777
Lotti	130 nm	14 x 14	132
Ingo	130 nm	19 x 19	58
Bill	130 nm	26 x 26	11
Joachim	180 nm	14 x 14	288

Table 2: Our testbed

	Relative objective values w. r. t.		
	Netlength	Yield(κ_1)	Yield(κ_2)
Netlength optimization	1.000	1.137	1.141
Yield(κ_1) optimization	1.109	1.000	1.265
Yield(κ_2) optimization	1.052	1.091	1.000

Table 3: Comparison of objective values on Monika

	Relative objective values w. r. t.		
	Netlength	Yield(κ_1)	Yield(κ_2)
Netlength optimization	1.000	1.200	1.420
Yield(κ_1) optimization	1.038	1.000	1.386
Yield(κ_2) optimization	1.028	1.146	1.000

Table 4: Comparison of objective values on Edgar

chips. The first objective is netlength (and via) minimization, where we set the $c^{\min}$ and $c^{\max}$ values to geometrical edge lengths (via edges have been set to a length of 12 routing tracks in our experiments). The other two objectives are yield optimization (i. e. critical area minimization), but with two different vectors κ_1 and κ_2 of defect weights that have been provided to us by manufacturing. We chose $s_{e,N} := 1$ for all $e \in E(G)$ and $N \in \mathcal{N}$ and computed $c^{\min}$ and $c^{\max}$ as described in section 5. The tables show relative objective values, i. e. objective values divided by the value obtained when optimizing the corresponding objective, so the diagonal values are all 1. These results show that critical area can increase considerably if optimizing only netlength or if not optimizing with correct weights for open and short defects.

While this already demonstrates the huge potential of yield optimization in global routing, our main focus is on quality of results after detailed routing. We compare our results after detailed routing to those obtained by using an implementation of Albrecht's global routing approach [1] which focuses on netlength and congestion only and is a two-dimensional approach, which means that all horizontal and vertical planes, respectively, are merged to a single plane, resulting in a smaller global routing graph. The two-dimensional global routing step is followed by a heuristic plane assignment step which tries to assign nets to routing planes which are good from a yield perspective. We summarize the results of this implementation in the "2D-GR" columns in tables 5 to 8.

In our approach, we have to see the different plane characteristics, so we have a three-dimensional global routing graph. This costs running time in global routing, but on the other hand can save time in detailed routing since search space can be restricted also in z dimension. The results obtained when using our global routing algorithm are shown in the "3D-GR" columns. All yield optimization experiments have been done with the κ_1 weights from above.

The results in tables 5 to 8 have been produced on an Opteron machine with 2.6GHz and 64 GB memory, using a single processor. Table 5 compares global routing running times of the different approaches. Compared to Albrecht's approach, our algorithm needs considerably more CPU time due to the larger global routing graph. However, Albrecht's approach cannot be parallelized efficiently, but our runtimes can be cut down drastically by parallelization, as was demonstrated in table 1. Using eight processors on a current machine, all chips in our testbed could be routed

Chip	2D-GR	3D-GR, Netl. Opt.		3D-GR, Yield Opt.	
Edgar	4,057	6,452	(+59.0%)	30,983	(+663.7%)
Hannelore	1,924	3,743	(+94.5%)	4,540	(+136.0%)
Paul	489	2,337	(+377.9%)	1,719	(+251.5%)
Monika	3,342	8,226	(+146.1%)	19,234	(+475.5%)
Garry	2,555	4,366	(+70.9%)	15,665	(+513.1%)
Heidi	2,162	2,900	(+34.1%)	8,831	(+308.5%)
Lotti	801	1,547	(+93.1%)	2,290	(+185.9%)
Ingo	569	2,124	(+273.3%)	2,471	(+334.3%)
Bill	1,298	835	(–35.7%)	2,279	(+75.6%)
Joachim	560	1,880	(+235.7%)	2,856	(+410.0%)
Total	17,757	34,410	(+93.8%)	90,868	(+411.7%)

Table 5: Global router CPU time (seconds on Opteron, 2.6 GHz)

Chip	2D-GR	3D-GR, Netl. Opt.		3D-GR, Yield Opt.	
Edgar	211.656	212.022	(+0.2%)	214.162	(+1.2%)
Hannelore	30.110	30.239	(+0.4%)	31.006	(+3.0%)
Paul	9.888	9.903	(+0.2%)	9.999	(+1.1%)
Monika	263.936	264.123	(+0.1%)	273.793	(+3.7%)
Garry	221.950	221.989	(+0.0%)	227.186	(+2.4%)
Heidi	150.775	150.863	(+0.1%)	153.837	(+2.0%)
Lotti	18.208	18.230	(+0.1%)	18.679	(+2.6%)
Ingo	13.199	13.285	(+0.7%)	13.482	(+2.1%)
Bill	23.312	23.356	(+0.2%)	23.542	(+1.0%)
Joachim	62.250	62.432	(+0.3%)	63.721	(+2.4%)
Total	1,005.284	1,006.442	(+0.1%)	1,029.407	(+2.4%)

Table 6: Wire length ([m], after detailed routing)

in less than two hours, both with netlength and with yield optimization.

Table 5 also shows that our algorithm is considerably slower if yield is optimized instead of netlength. There are two reasons for this: First, our experiments show faster convergence for netlength optimization than for yield optimization. In the first case, good results are obtained already after 10 to 20 phases, while for yield optimization typically 40 to 50 phases are needed (however, this is still much better than what can be guaranteed theoretically, cf. [12]). The second reason is that due to the differences between defect sensitivities on different routing planes, future cost quality degrades significantly: Future cost must always be a lower bound, but not all nets can be routed on the best possible plane, of course. Also in detailed routing, future cost is essential for obtaining good running times. Future cost computation in our detailed router does not take into ac-

Chip	2D-GR	3D-GR, Netl. Opt.	3D-GR, Yield Opt.
Edgar	6,151,607	6,114,859 (–0.6%)	8,302,895 (+35.0%)
Hannelore	795,855	804,856(+1.1%)	1,096,198 (+37.7%)
Paul	474,376	449,112 (–5.3%)	606,733 (+27.9%)
Monika	9,335,637	8,916,882 (–4.5%)	12,409,600 (+32.9%)
Garry	6,018,048	5,740,090 (–4.6%)	8,555,230 (+42.2%)
Heidi	5,030,429	4,790,479 (–4.8%)	6,821,014 (+35.6%)
Lotti	669,582	649,336 (–3.0%)	797,861 (+19.2%)
Ingo	441,647	429,608 (–2.7%)	586,823 (+32.9%)
Bill	103,812	101,471 (–2.3%)	185,742 (+78.9%)
Joachim	1,924,130	1,937,133(+0.7%)	2,026,975 (+5.3%)
Total	30,945,123	29,933,826 (–3.3%)	41,389,071 (+33.7%)

Table 7: Number of vias (after detailed routing)

Chip	2D-GR	3D-GR, Netl. Opt.		3D-GR, Yield Opt.	
Edgar	0.09780	0.10493	(+7.3%)	0.08586	(–12.2%)
Hannelore	0.01396	0.01543	(+10.6%)	0.01027	(–26.4%)
Paul	0.00502	0.00568	(+13.2%)	0.00402	(–19.9%)
Monika	0.08744	0.09505	(+8.7%)	0.08055	(–7.9%)
Garry	0.07224	0.08017	(+11.0%)	0.06714	(–7.1%)
Heidi	0.05351	0.05804	(+8.5%)	0.04965	(–7.2%)
Lotti	0.00658	0.00688	(+4.5%)	0.00575	(–12.6%)
Ingo	0.00457	0.00505	(+10.4%)	0.00392	(–14.2%)
Bill	0.00707	0.00833	(+17.8%)	0.00376	(–46.8%)
Joachim	0.00432	0.00440	(+1.9%)	0.00431	(–0.1%)
Total	0.35251	0.38396	(+8.9%)	0.31523	**(–10.6%)**

Table 8: Expected number of faults per chip (after detailed routing)

count detours prescribed by global routing, so path search takes more time if global routing has been done with the objective of optimizing yield because many nets are routed with some detours to obtain better spreading.

This is also reflected in table 6, and table 7 shows that also the number of vias goes up significantly when optimizing yield. This seems contrary to the fact that vias generally are considered harmful for manufacturing yield, but it turns out that with the open and short defect weights used in our experiments, it is worthwhile to spend extra vias in order to put more wiring on higher planes. Table 8 shows the estimated number of faults per chip in the wiring (including vias) after detailed routing. Note that the results in the first column are better than in the second one because of the heuristic plane assignment step. The third column shows that our approach, which provably finds a near-optimum solution, can again drastically improve results compared to the first column, in average by more than 10 percent.

Note that exactly the same detailed router with the same parameters was used for all our experiments. By adapting the detaild router optimally one can certainly improve the yield even further.

7. CONCLUSION

We have presented an enhanced global routing approach and have shown how to use it for yield optimization. We have run it on several state-of-the-art VLSI routing instances from industry and showed that our algorithm reduces the expected number of faults significantly, in average by more than 10 percent. We can use it to considerably improve yield. Our algorithm is parallelized very efficiently and can route even the largest chips in our testbed within less than two hours.

Our algorithm can also be used to optimize power consumption and to limit capacitance on critical paths in order to avoid timing violations after routing. Further experiments have to be done to demonstrate these capabilities on VLSI routing instances.

Moreover, we did not consider at all yield optimization in detailed routing in this paper. Certainly detailed routing offers further potential for significant yield improvements.

8. ACKNOWLEDGEMENT

First of all, I would like to thank Jens Vygen for many helpful discussions and suggestions. I also want to thank the anonymous referees for their very useful comments.

9. REFERENCES

[1] C. Albrecht, “Global routing by new approximation algorithms for multicommodity flow”, in IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 20, pp. 622–632, 2001.

[2] R. C. Carden IV, J. Li and C.-K. Cheng, “A global router with a theoretical bound on the optimum solution”, in IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 15, pp. 208–216, 1996.

[3] S. E. Dreyfus and R. A. Wagner, “The Steiner problem in graphs”, in Networks, vol. 1, pp. 195–207, 1972.

[4] N. Garg and J. Könemann, “Faster and simpler algorithms for multicommodity flow and other fractional packing problems”, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, pp. 300–309, 1998.

[5] T.-Y. Ho, Y.-W. Chang, S.-J. Chen and D.-T. Lee, “A fast crosstalk- and performance-driven multilevel routing system”, in Proceedings of the IEEE international Conference on Computer-Aided Design, Nov. 2003.

[6] J. Hu and S. S. Sapatnekar, “A survey on multi-net global routing for integrated circuits”, in Integration, the VLSI Journal, vol. 31, pp. 1–49, 2001.

[7] T. Jing, X. Hong, H. Bao, Y. Cai, J. Xu, C. Cheng and J. Gu, “Utaco: A unified timing and congestion optimizing algorithm for standard cell global routing”, in Proceedings of the Asia and South Pacific Design Automation Conference, pp. 834–839, 2003.

[8] W. Maly, “Modeling of lithography related yield losses for CAD of VLSI circuits”, IEEE Transactions on Computer-Aided Design, vol. CAD-4, pp. 166-177, July 1985.

[9] P. Raghavan, “Randomized rounding and discrete ham-sandwich theorems: provably good algorithms for routing and packing problems”, Ph.D. thesis, Report No. UCB/CSD 87/312, University of California, Berkeley, 1986.

[10] P. Raghavan and C. D. Thompson, “Randomized rounding: a technique for provably good algorithms and algorithmic proofs”, in Combinatorica, vol. 7, pp. 365–374, 1987.

[11] F. Shahrokhi and D. W. Matula, “The maximum concurrent flow problem”, in Journal of the ACM, vol. 37, pp. 318–334, 1990.

[12] J. Vygen, “Near Optimum Global Routing with Coupling, Delay Bounds, and Power Consumption”, in Integer Programming and Combinatorial Optimization; Proceedings of the 10th International IPCO Conference; LNCS 3064 (G. Nemhauser, D. Bienstock, eds.), pp. 308-324, Springer, Berlin 2004,

[13] J. Xu, X. Hong, T. Jing, Y. Cai and J. Gu, “A novel timing-driven global routing algorithm considering coupling effects for high performance circuit design”, in Proceedings of the Asia and South Pacific Design Automation Conference, pp. 847–850, 2003.