METHODS AND ALGORITHMS FOR COUPLING REDUCTION

By

Ryan Kastner

A Thesis Submitted to the Graduate

Faculty of Northwestern University

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Approved:

Professor Majid Sarrafzadeh
Thesis Adviser

Northwestern University
Evanston, Illinois

August 21, 2000

CONTENTS

LIST OF TABLES

LIST OF FIGURES

To my family (Dad, Mom, Erin and dot)

ABSTRACT

In this work, we develop methods to reduce interconnect delay and noise caused by coupling. First, we introduce a novel problem that deals with coupling – the Coupling-Free Routing (CFR) Problem. CFR takes a set of nets and tries to find a one-bend routing for a subset of nets. A routed net must not couple with any other routed net. We argue that this problem is useful in both global and detailed routing. It guarantees that the routing has minimum delay since the wirelength, number of vias and coupling are minimized.

Then, we present an exact algorithm for the CFR decision problem via a transformation to 2-satisfiability. The decision problem determines whether the given set of nets is coupling-free routable.

Next, we introduce a new type of graph, the implication graph which models the dependencies associated with CFR. Also, we look at some of the properties associated with the graph.

Finally, we develop three algorithms to efficiently solve the Maximum Coupling-Free Layout (MAX-CFL) problem. Given a set of nets, the problem is defined as finding a subset of nets that are coupling-free routable. The subset should have maximum size and/or criticality. The criticality of a net can be defined in a variety of ways e.g. timing slack, wirelength, etc.. The first algorithm, the greedy algorithm, is a simple, yet effective method for finding a set of nets which have maximum criticality. The direct forcing algorithm uses the notion of forcing to find a solution to the MAX-CFL problem. The final algorithm we present is the implication algorithm. It uses properties associated with the implication graph and consistently finds the best solution in terms of number of nets routed. Our experimental results show that the algorithms work effectively on real data.

1. INTRODUCTION

1.1 Coupling

As fabrication technology moves into deep sub-micron (DSM) device sizes and gigahertz clock frequencies, interconnect becomes an increasingly dominate factor in performance, power, reliability and cost [56]. In particular, coupling is of greater importance for power, area and timing in circuits [38]. There are four principal reasons for this, increasing interconnect densities, faster clock rates, more aggressive use of high performance circuit families, and scaling threshold voltages [37].

In order to keep wiring resistance from increasing too quickly, many processes are scaling the wire height at a slow rate compared to wire pitch. This results in taller, thinner wires. Also, spacing between wires is decreasing in order to yield high packing densities. A detrimental side effect of these scaling trends is an increased amount of coupling capacitance [10]. Coupling capacitance is proportional to the amount of parallel overlap between the wires and inversely proportional to the distance between the wires (coupling is formally defined in Chapter 2). In fact, coupling capacitance between wires can account for over 70% of the total wiring capacitance, even in 0.25 μm processes [11]. There are two problems introduced by coupling, delay deterioration and crosstalk.

Delay deterioration refers to the fact that the total capacitance seen by a gate is no longer a constant value [13, 17]. The rising contribution of coupling capacitance to total load capacitance makes the Miller effect evident. The Miller effect says that if both terminals of a capacitor are switched simultaneously, the effective capacitance between the terminals is modified [22]. Assume that the coupling capacitance between two wires is C_c. If the two wires switch in opposite directions (e.g. one wire goes from high to low, the other goes from low to high), the equivalent coupling is

$2C_c$. On the other hand, an anti-Miller effect will occur when the wires switch in the same direction. Here, the equivalent coupling is 0. As you can see, the coupling effect is not always detrimental. Delay deterioration occurs because the Miller effect causes the capacitance to vary. For example, if two coupled nets switch in opposite directions at the same time, the capacitance, hence the delay, will increase.

Crosstalk is a type of noise[1] introduced by coupling between two adjacent wires. A change in voltage or current on one of the wires may interfere with the signal on the other wire. There are two unwanted effects of crosstalk. First, the two wires form a mutual inductor. This inductive effect must be considered as circuit frequencies move above 500 MHz [42, 64]. Inductive effects are not addressed this work. The second effect is associated with coupling capacitance. Coupling capacitance can cause a switching net to induce noise onto a neighboring net possibly resulting in an incorrect functional response.

1.2 Routing

The circuit layout problem is partitioned into a collection of problems. There are two major problems, placement and routing. The placement problem is related to several problems: partitioning, floorplanning and placement [44].

1. *Circuit partitioning* is the task of dividing the circuit into smaller parts.
2. *Floorplanning* determines the approximate location of each module in a rectangular chip area.
3. *Placement* takes a set of modules and a net list and provides a position for each module according to a cost function. The two prevalent cost functions are wire-length based and cut-based.

[1]Noise is defined as an unwanted variation which makes the behavior of a manufactured circuit deviate from the expected response [30].

The process of routing can be divided into two subproblems, global and detailed routing. Global routing decomposes the routing problem into smaller, manageable routings for the detailed router. Specifically, the global router finds a rough path for each net while trying to reduce the chip size, shortening the wire length and distributing the congestion across the routing area, among other things [44, 47, 59]. Detailed routing uses the results of global routing to find an exact realization of the interconnections in VLSI circuits.

The global routing problem is known to be NP-hard [54]. This motivates the use of heuristic and approximation algorithms. The maze routing (or maze running) algorithm [7] is a widely used method for global and detailed routing [12, 25, 39, 41, 48, 50]. Briefly, the maze routing algorithm starts from a source point and recursively searches it's neighbors for the best route until it reaches the sink point. The best route is defined by a function of congestion, wire length, chip size, number of bends, etc.. Maze routing finds the optimal path for two-terminal nets according to the cost function. A major drawback of the algorithm is the large amount of memory required to label it data structure, the grid graph.

A goal-directed unidirectional search heuristic [49] (also known in AI as A*) can improve the run time of the maze router. This heuristic adds the length from the source and the estimated length to the sink to the cost function. This allows the most direct routes from source to sink to be searched first. Most often, the actual routing closely resembles the direct route. This is the motivation for the heuristic and often allows a run time improvement with little sacrifice in the quality of the routing solution.

There have been several other proposed extensions and modifications to the maze routing algorithm in the almost 40 years since it has been introduced, but the underlying method remains the same.

Most existing routing tools consider wirelength, congestion and routability; up until recently, there has been little research on the coupling problem in routing. Coupling reduction was considered at the detailed routing stage for the river routing problem [20], the channel routing problem [34] and the switchbox routing problem [58]. Also, there have been efforts in reducing coupling in the stage between global and detailed routing. Xue *et al.* developed a post global routing tool which estimates the possible coupling between sensitive wires and tries to reroute nets away from possible crosstalk areas [60]. Chaudhary *et al.* develop a general post-routing spacing algorithm [33]. Also, coupling is examined for area routing [23] and global routing [21]. This work presents algorithms for coupling avoidance routing. The algorithms are general so they can be used in both global and detailed routing.

In this work, we focus on reducing the unwanted effects caused by coupling during routing. In Chapter 2, we give some basic definitions, formally define coupling and introduce the Coupling-Free Routing (CFR) Problem. An exact algorithm for Coupling-Free Routing Decision Problem (CFRDP) is given Chapter 3. A new type of graph, the implication graph, is introduced in Chapter 4. An implication graph models the dependencies between nets in the CFR problem. Chapter 5 introduces the Maximum Coupling-Free Layout (MAX-CFL) Problem and analyzes several algorithms developed to solve the problem. We conclude in Chapter 6.

2. PRELIMINARIES

A *multi-terminal net* n $= \{(x_1, y_1,), (x_2, y_2), (x_3, y_3), ..., (x_n, y_n)\}$ is a collection of points in the plane. A *terminal* is single point of a net. A multi-terminal net can be partitioned into a collection of *two-terminal* nets (a net with exactly two points) using a number of standard techniques. We shall adopt the *stable spanning tree* partitioning of Ho *et al.* [29].

A two-terminal net (or simply called a *net* hereafter) $n = \{(x_1, y_1), (x_2, y_2)\}$ is an unordered pair of points (x_1, y_1) and (x_2, y_2). A *routing* or *wiring* of n is a set of horizontal and vertical line segments connecting (x_1, y_1) and (x_2, y_2). A *layout* is the routings of a set of nets.

A net n can be routed without any bends if and only if either $x_1 = x_2$ or $y_1 = y_2$. We call such a net a *zero-bend net.* Otherwise, there are two ways to route n with one bend as shown in Figure 2.1. When a routing has no more than one bend, it is called a *single bend routing* [61]. We call such a net a *one-bend net.*

The routings in Figure 2.1 are called the *upper-L routing* and the *lower-L routing.* A stable spanning tree ensures that upper-L routing and lower-L routing shapes of the two-terminal nets obtained from a multi-terminal net are pairwise non-intersecting. To avoid confusion, we often refer to a possible routing as a *route.* Thus we say that a one-bend net has two one-bend routes (the upper-L route and the lower-L route).

Given a multi-terminal net N, a *rectilinear Steiner tree* (RST) $T(V, E)$ is a spanning tree in the rectilinear plane that connects every node in $V = T \bigcup I$, where T is the set of terminals for the given net N and I is a set of internal Steiner nodes of degree three or four. For example, the RST in Figure 2.2 shows a Steiner tree with terminals t_1, t_2, t_3, t_4, t_5 and Steiner nodes s_1, s_2.

A *Minimum Steiner tree* (MST) is a RST with minimum length. The general

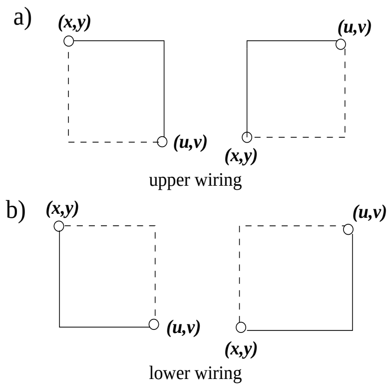

Figure 2.1: a) Upper-L routings b) Lower-L routings

MST problem is NP-Hard [18]; only restricted classes of the problem can be solved in polynomial time. There are many exact and heuristic algorithms for solving the problem [3, 5, 9, 16, 31, 32, 43].

A RST is stable if there is no pair edges whose enclosing bounding boxes intersect or overlap, except touching at a common end point (if any) of the two segments [29]. Referring to Figure 2.2, the Steiner tree in a) is an unstable RST; flipping the (t_4, s_2) edge to a lower-L routing creates overlap with the edge (t_3, s_2). Figure 2.2 b) is a stable RST. If we have a stable RST, we can partition the multi-terminal net into two-terminal nets and route the two-terminal nets in a L-shape pattern without worrying about overlap between the edges of the multi-terminal net.

We focus on routing *critical nets*. Critical nets can be defined in a variety of ways. Most often, critical nets correspond to nets that are on a critical path of a network at the logic synthesis stage [15]. Interconnect delay of the these nets should be minimal, therefore we are interested in a single bend routing. A single bend routing is not only the shortest possible route between the two terminals, it also introduces only one via. Since vias further increase the wire capacitance and resistance, it is beneficial to keep them at a minimum. Also, vias negatively effect the routability of the circuit [4].

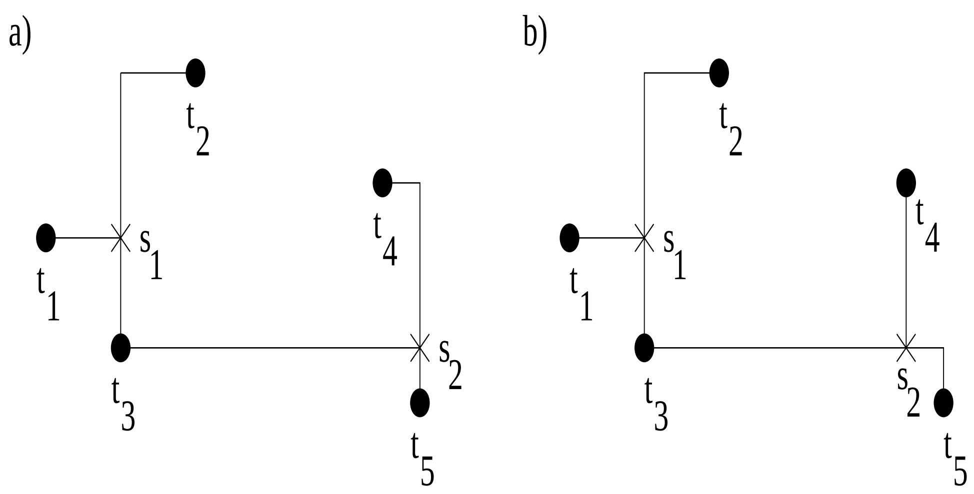

Figure 2.2: a) Unstable RST b) Stable RST

2.1 Coupling

Bakoglu [22] shows that the wire-delay on a distributed RC line contains a $R_W \cdot (C_S + C_C)$ time constant, where R_W is the interconnect resistance, and C_S and C_C are the substrate and coupling capacitances.

$$R_W \cdot (C_S + C_C) = \frac{\rho \cdot l}{w \cdot t}\left(\frac{\varepsilon_\kappa \cdot l \cdot w}{h} + \frac{\varepsilon_\kappa \cdot l \cdot t}{s}\right) \tag{2.1}$$

where ρ is resistivity of the conductor, ε_κ is the insulator dielectric constant, and w, t and h are the conductor's width, thickness and separation from the substrate, respectively. The terms l and s are the coupled length and spacing of the interconnect.

The coupling capacitance C_C between two wires i and j can also be represented as follows:

$$C_C(i,j) = \frac{f_{ij} \cdot l_{ij}}{d_{ij}} \frac{1}{1 - \frac{w_i + w_j}{2d_{ij}}} \tag{2.2}$$

where w_i and w_j are the sizes of wires i and j ($w_i, w_j > 0$), f_{ij} is the unit length fringing capacitance between wires i and j, l_{ij} is the overlap length of wires i and j and d_{ij} is the distance from the center line of wire i to the center of wire j (see

Figure 2.3).

We are trying to minimize the coupling. During routing, we can control l_{ij}, d_{ij}, w_i and w_j. By avoiding overlap between two wires, l_{ij} can be minimized. In other words, we do not want adjacent wires to run in parallel for long distances. We assume that w_i, w_j, l_{ij} are fixed; we do not consider wire sizing and spacing in our algorithm. But, this can be done as a post-processing step using a number of techniques (see [26, 27] for a comprehensive survey and tutorial).

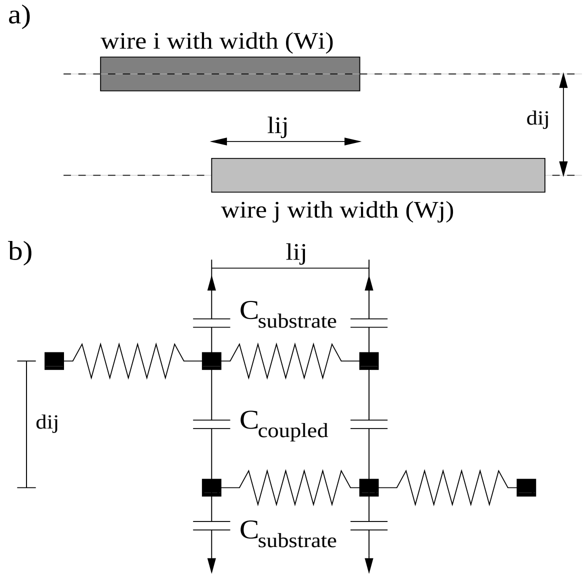

Figure 2.3: a) Physical coupling capacitance between two wires b) The wires modeled by resistors and capacitors

Noise analysis and avoidance is becoming increasingly critical in the design of digital circuits. Performance demands often lead to the use of dynamic logic circuit families and its derivatives. Dynamic circuits use a clock signal (or something similar) to precharge the output voltage. The circuit is evaluated solely through the use of tree structure of NMOS transistors. The advantage is that the capacitance seen by the previous stage is reduced. The driving transistor has to drive the capacitance of the NMOS transistor; CMOS logic necessitates the need to drive a NMOS and PMOS transistor. The disadvantages are the precharge time and the reduced switching point. CMOS circuits have a switching point around half the supply voltage. The switching point of dynamic logic is the threshold voltage of the NMOS transistor. Hence, these aggressive dynamic circuit families trade noise margin for timing performance. They are more susceptible to noise failure which increases the need for noise analysis.

The problem of noise failure can be more severe than timing failure. Timing failure can be recovered by increasing the clock speed which allows more time for signal propagation. Noise is caused by capacitive coupling and input slope, among other things. Such things are difficult, if not impossible, to control from the chip terminals. If a noise problem is undetected correcting it may cause an expensive second fabrication run [2].

Crosstalk (coupling noise) impacts digital circuits by introducing logic and delay hazards. For example, asynchronous lines and precharge structures are especially susceptible to crosstalk glitches. Since increasing the cycle time will not alleviate such hazards, these are static or logical hazards. Coupling between nets with close timing can induce additional delay by injecting charge on the more weakly driven node, and potentially cause a delay hazard.

A delay hazard occurs when two nets that are tightly coupled have closely timed opposing transitions. The strongly-driven net called the *aggressor* will inject charge on the weakly-driven net or *victim*. Looking at Figure 2.4, the signals on nets A and B can not cause a delay fault unless they switch nearly simultaneously. Figures 2.4 a) and b) show only a small glitch on net A since the respective signals transition separately. Figure 2.4 c) shows that both nets experience additional delay which could lead to a delay hazard.

In [8], Kirkpatrick and Sangiovanni-Vincentelli introduce the notion of *crosstalk constraint generation* which uses the concepts of *analog and digital sensitivity* and a physical coupling term in order to reduce the constraints given to layout synthesis.

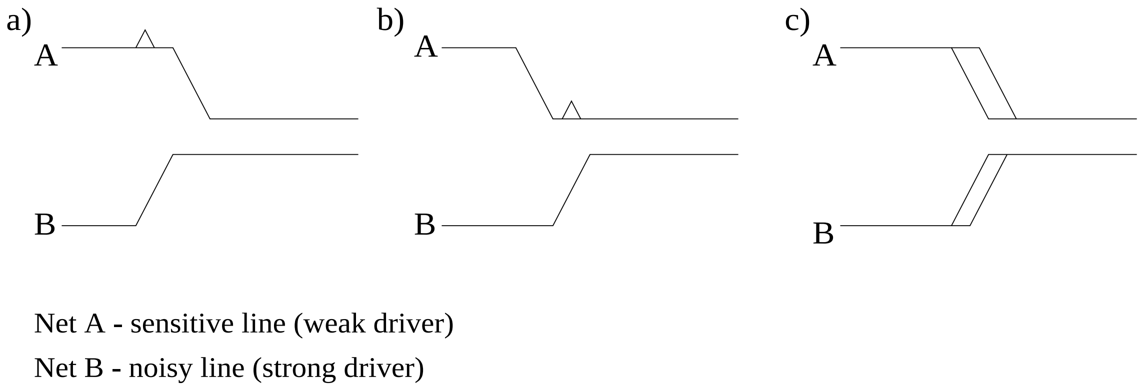

Figure 2.4: A delay fault occurs only when two signals have opposing transitions and switch simultaneously

This allows us to remove false crosstalk constraints. For example, a net A may couple with net B. But, net A could have a high tolerance for delay and noise. Therefore, the A and B can couple without negative circuit performance. We want to remove these cases as they unnecessarily over-constrain the problem.

In most cases, the crosstalk voltage amplitude determines circuit failure. With a large enough amplitude, the transistor will cross it's threshold value causing a static hazard (as in the case of domino logic) or a delay hazard by causing an extra charging delay. The noise voltage bound on a net i and its sensitivity to switching on another net j is defined as:

$$\Delta V_i \leq \Delta V_{Bi} \tag{2.3}$$

$$S_{ij} = \frac{\delta V_i}{\delta V_j} \tag{2.4}$$

Since we are dealing with digital circuitry, we can normalize all noise sources j to the power rail swing: $\Delta V_j = \Delta V$.

The analog sensitivity deals with basic circuit interaction e.g. drive strength. Most often, circuit simulation techniques (see [19]) are used to obtain the analog sensitivity factor for net i, A(i). The digital sensitivity, D(i), includes temporal and logical separation of net interaction. Net i is digitally sensitive to Net j only if they transition in opposite directions (logical) within a certain time window (temporal).

The physical coupling is defined by a capacitive divider relation:

$$Q(i,j) = \frac{C(i,j)}{C_{Ti}} \tag{2.5}$$

where $C(i,j)$ is the coupling capacitance between nets i and j and C_{Ti} is the total capacitance of net i.

Finally, the sensitivity between nets i and j can be redefined as:

$$S_{ij} = A(i,j)D(i,j)Q(i,j) \tag{2.6}$$

Then, every net i is given a bound $B(i)$ which relates to the tolerance of that net to noise, delay, etc.. A large bound value indicates that the net has a high tolerance. The crosstalk constraints of *net* $i, N(i)$, are the sum of the sensitivities with respect to the other nets.

$$N(i) = \sum_j S_{ij} \tag{2.7}$$

A *critical net* is defined as a net that exceeds it allowable bound i.e. $N(i) > B(i)$.

2.2 Pattern Routing

Pattern routing is the notion of using predefined patterns to route two-terminal nets. Usually these are simple patterns such as a L-shaped (sometimes called 1-bend) or a Z-shaped pattern – 2-bends, route restricted within bounding box. For more details, see Figure 2.5.

A *grid graph* is a graph G(V,E) such that each vertex corresponds to a point in a plane. See Figure 2.6 for further explanation. A *routing* of a net on a grid graph is a set of grid edges such that the terminals are fully connected. The *route*

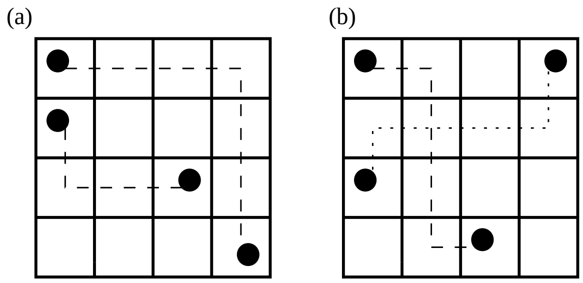

Figure 2.5: (a) L-shaped routing of 2 two-terminal nets. (b) Z-shaped routing of 2 nets.

edges of a net are the set of edges used in the routing of that net.

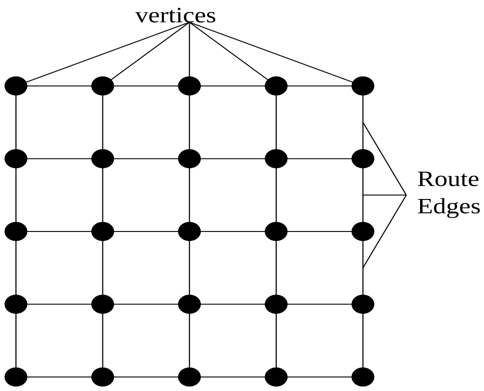

Figure 2.6: A grid graph.

Patterns can speed up the routing process. Instead of maze routing a net, we pattern route it. In general, maze routing will consider many bins that the final route will not actually use. When using pattern routing, only a constant number of edges are searched. For example, L-shaped pattern routing will only search the edges on the bounding box of the two-terminal net. Then, depending on cost of these edges, it will choose the upper-L or lower-L and place the route there. Similarly, Z-shaped pattern routing needs only search the edges on the perimeter and inside the two-terminal bounding box. On the other hand, maze routing will search every edge (on the worst case). Therefore, pattern routing has a better upper bound on runtime complexity. We found that on average, the pattern routing approach searches fewer edges than the maze router. We formally summarize the complexities:

1. Given a net $n = \{(x_1, y_1), (x_2, y_2)\}$ and a grid graph G(V,E).

2. Let A be the edges on and within the bounding box of n. A $\subseteq$ E. $|A| = 2 \cdot |x_1 - x_2| \cdot |y_1 - y_2| + |x_1 - x_2| + |y_1 - y_2|$

3. Let P be the edges on the bounding box of n. P $\subseteq$ A. $|P| = 2 \cdot (|x_1 - x_2| + |y_1 - y_2|)$

4. Maze routing - $O(|E|)$

5. L-shaped pattern routing - $O(|P|)$

6. Z-shaped pattern routing - $O(|A|)$

Theorem 2.1: $|P| \leq |A| \leq |E|$.

Proof: The proof is trivial since, by definition, $P \subseteq A \subseteq E$. □

The maze router ensures that the least cost route (according to the cost function) is found. Pattern routing does not give you this luxury. In fact, an L-shaped pattern routing could produce the second worst possible route. This occurs if both the upper-L route and the lower-L route are the two worst paths. Pattern routing will choose the better of these two solutions, giving you a bad routing. In general this is not the case, as our results show.

Another benefit of pattern routing lies in the predictability of a pattern-routed net [51]. If you know that a net will be pattern routed, you can quickly and accurately estimate its route earlier in the design flow. For example, you know that an L-shaped pattern route will take one of two routes. This allows higher level CAD tools, such as the placement or logic synthesis engines, to estimate routings which will lead to better congestion and area estimates. In order to exploit predictability, the tools need placement information. Many industrial logic synthesis tools are moving towards layout-driven synthesis. Additionally, an academic behavioral level synthesis tool has recently incorporated placement information [62].

With emergence of deep sub-micron (DSM) fabrication technology, interconnect has an increasingly dominant role. Now circuit delay is determined by the gate resistance and capacitance as well as the interconnect resistance and capacitance [10]. When optimizing for delay in a circuit, logic synthesis tools look at the critical path. Usually these tools only consider the gate delay, ignoring the interconnect delay. If we could pattern route the gates on the critical path, then we can more accurately estimate the interconnect resistance and capacitance.

Finally, the number of vias on a pattern-routed net is fixed. Since vias further increase the capacitance and resistance, it is beneficial to keep them at a minimum. Also, vias negatively effect the routability of the circuit [4].

2.3 Coupling-Free Routing

Every route consists of horizontal and/or vertical line segments. We say two wires *couple* if the line segments forming them are closer than d units for more than l units. Two line segments *intersect* if they have at least one point in common and *overlap* if they have more than one point in common.

For a given set of nets $S = \{N_i = \{(x_{1i}, y_{1i}), (x_{2i}, y_{2i})\} \mid 1 \leq i \leq n\}$, a (single bend) layout of S is coupling-free if there are no two routes that run in parallel at a distance equal to or closer than d units for more than l continuous units. Examples of coupled and non-coupled layouts are given in Figure 2.7. Given a set of two-terminal nets, the problem of obtaining a coupling-free routing of nets is called the *coupling-free routing problem* (CFR problem) [52].

We feel that CFR is beneficial in both detailed and global routing.

As VLSI fabrication technology progresses, more routing layers become available. Therefore, we can afford to set aside *preferred layers* for critical nets. A

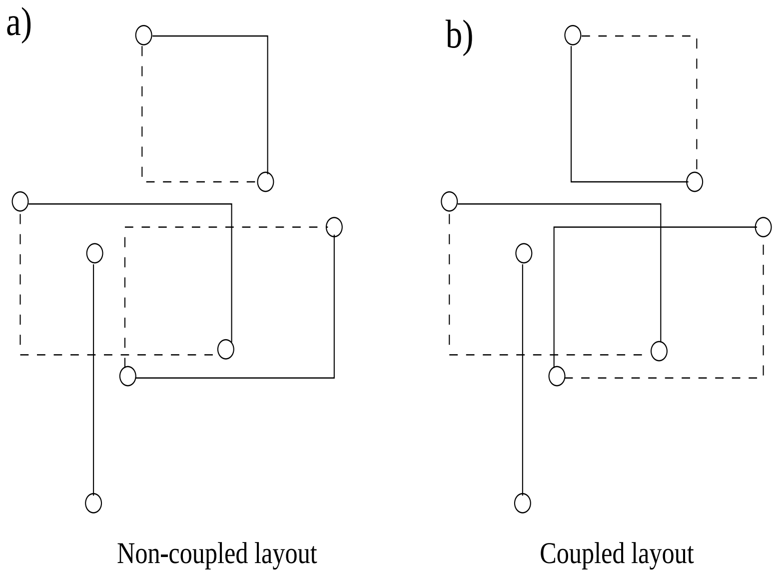

Figure 2.7: a) Coupling-free routings b) Non-coupling-free routings

preferred layer usually has a lower wiring resistance due to position of the layer (lower layers have lower resistance) and width of the wires on that layer (large wire widths have lower resistance). Power, ground and clock nets are already routed on preferred layers. We propose using the preferred layers for routing critical nets. Critical nets are allotted very little slack in order to meet timing constraints. Since interconnect is becoming a dominate factor in delay of a circuit and coupling plays are large role in interconnect delay, these nets should routed in order to minimize coupling and wirelength. Therefore, we can use notion of coupling-free routing to provide a detailed routing for the critical nets. Since the nets are routed with at most one bend, they have minimum wirelength. In addition, coupling-free routing minimizes the coupling of the routed nets. Combining these two factors, we have a routing of the critical nets with minimal interconnect delay. After we have a coupling-free layout, non-critical nets can be routed on the preferred layers to maximize routing resources.

Many single-layer routing algorithms have been suggested. Liao *et. al* [36] propose density routing or maze routing to perform this task. A more recent paper by Lin and Ro [65] improves on the work by Liao *et. al.* They employ a two step process. First, they find a planar set of single-bend nets. Then, they use a method based on rubber-band equivalent to find a routing for the remaining nets. CFR can easily be incorporated into the first stage of Lin and Ro's algorithm to obtain a planar layout that is coupling-free.

Generally, coupling at the global routing stage is hard to determine. A global route is not exact. Therefore, a net could possibly couple with every net that is routed in the same global bin. But, the net will only couple with it's two neighbors[2]. Ultimately, track assignment (which can be done at the global or detailed routing stage) determines the coupling. Additionally, the detailed router will often make local changes which can effect the coupling of nets [28]. But, the detailed router can only make local changes, therefore considering coupling at the global stage, even if it isn't exact, is beneficial as it can provide a way to make large scale changes to a layout that otherwise can not be done at the detailed level. If we have coupling-free layout at the global stage, then the layout will remain coupling-free at the detailed stage. Therefore, we can use CFR at the global routing stage to minimize coupling for the detailed router.

In Chapter 3, we propose an exact algorithm for determining if a set of nets can be a coupling-free routing. Then, in Chapter 5 we describe a couple heuristics for solving the *maximum coupling-free layout problem* – the maximum number of nets that can be laid out in a coupling-free fashion.

[2]Theoretically, a net couples with every net on the chip. But, the neighboring nets act as a shield which makes the coupling capacitance seen by the other nets minimal.

3. THE COUPLING-FREE ROUTING DECISION PROBLEM

The Coupling-Free Routing Decision Problem (CFRDP): Given a set of two-terminal nets S. Is there a single-bend routing for every net in S such that no two routings couple? That is, does there exists no two routes that run in parallel at a distance equal to or closer than d units for more than l continuous units?

We solve the coupling-free routing decision problem by transforming it into an instance of the 2-satisfiability (2SAT) problem.

The 2-satisfiability problem: Given a set U of variables, a collection C of clauses such that each clause $c \in C$ has $|c| = 2$. Is there a satisfying truth assignment for U? The 2SAT problem can be solved in $O(|U|)$ time [55].

In order to transform an instance of CFR decision problem to 2SAT, we assign a boolean variable to each net. Without loss of generality, we say if net A has an upper-L route if it's variable is true (x_A) and a lower-L route if it's variable is false ($\overline{x_A}$). A routing of a net may *force* a routing of another net. For example, assume net A is routed in an upper-L. If the upper-L routing of A (x_A) couples with the lower-L routing of B ($\overline{x_B}$), then net B must be routed as an upper-L to avoid coupling. Hence x_A forces x_B. With respect to two nets A and B, there are 10 possible forcing interactions between these nets.

1. A and B are independent. Either layout for each net does not directly influence the layout for the other.

2. A and B can not be couple-free routed.

3. The lower-L routing for A forces the upper-L routing for B. However, the upper-L routing for A does not influence the routing of B. The next three

cases are similar.

4. The lower-L routing of A forces the lower-L routing of B.

5. The upper-L routing of A forces the upper-L routing of B.

6. The upper-L routing of A forces the lower-L routing of B.

7. The lower-L routing of A forces a lower-L routing of B. Also, the upper-L routing of A forces an upper-L routing of B. The next three cases are similar to this case.

8. Lower-L of A forces lower-L of B; Upper-L of A forces lower-L of B.

9. Lower-L of A forces upper-L of B; Upper-L of A forces upper-L of B.

10. Lower-L of A forces upper-L of B; Upper-L of A forces lower-L of B.

Examples of all of these cases are given in Figure 3.1.

The algorithm proceeds as follows:

Stage 1: Consider the $\frac{n(n-1)}{2}$ interactions (where $n = |S| =$ number of nets) between the nets. If two nets can not be couple-free routed (corresponding to interaction 2), the algorithm terminates and returns FALSE. For each pair of nets, i and j, we determine the interaction between N_i and N_j. Using this information, we can determine which wires are forced.

Stage 2: The constraint information must be encoded into boolean expression with these properties:

1. It is in conjunctive normal form (CNF) (see [40])

2. It contains at most two literals per clause

3. It is satisfiable if and only if the corresponding wire set can be laid out (without coupling) in a single bend fashion

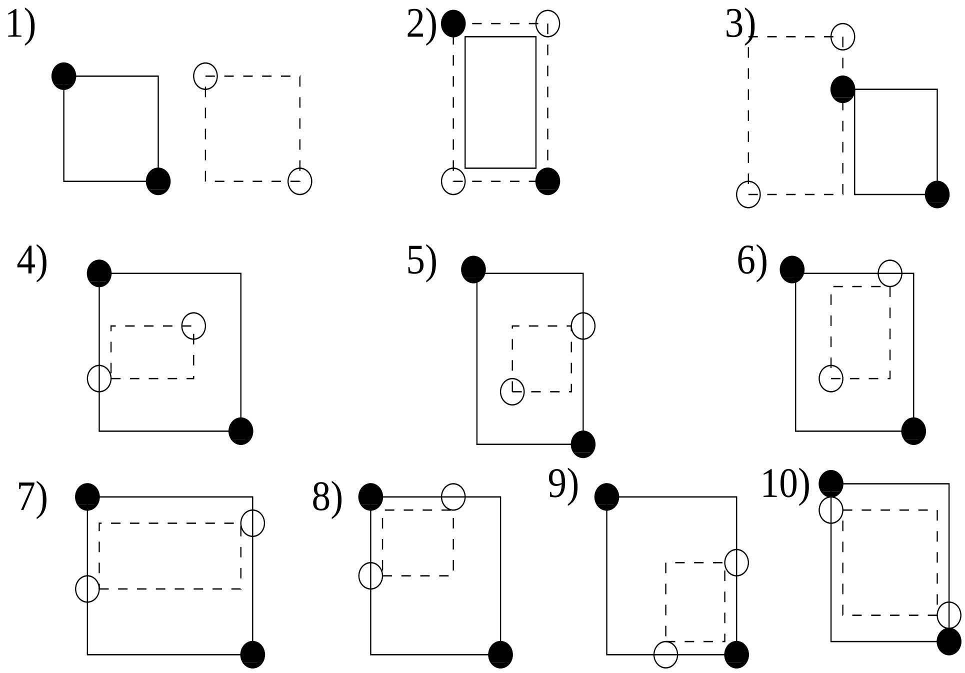

Figure 3.1: Examples of the 10 interactions for the coupling-free routing problem. The solid points and lines correspond to net A. The dotted lines and circles correspond to the the bounding box and terminals of net B, respectively.

Each of the 10 interactions can be encapsulated as a binary relation.

1. A and B are independent. No encoding

2. A and B can not be couple-free routed. No encoding, the algorithm will terminate and return FALSE if this case is found.

3. The lower-L routing for A forces the upper-L routing for B. Encoded as $(x_A \vee x_b)$

4. The lower-L routing of A forces the lower-L routing of B. Encoded as $(x_A \vee \overline{x_B})$

5. The upper-L routing of A forces the upper-L routing of B. Encoded as $(\overline{x_A} \vee x_B)$

6. The upper-L routing of A forces the lower-L routing of B. Encoded as $(\overline{x_A} \vee \overline{x_B})$

7. The lower-L routing of A forces a lower-L routing of B. Also, the upper-L routing of A forces an upper-L routing of B. Encoded as $(x_A \vee \overline{x_B}) \wedge (\overline{x_A} \vee x_B)$

8. Lower-L of A forces lower-L of B; Upper-L of A forces lower-L of B. Encoded as $(x_A \vee \overline{x_B}) \wedge (\overline{x_A} \vee \overline{x_B})$

9. Lower-L of A forces upper-L of B; Upper-L of A forces upper-L of B. Encoded as $(x_A \vee x_B) \wedge (\overline{x_A} \vee x_B)$

10. Lower-L of A forces upper-L of B; Upper-L of A forces lower-L of B. Encoded as $(x_A \vee x_B) \wedge (\overline{x_A} \vee \overline{x_B})$

For each forced wire A, if the wire is forced to an upper-L route, this is encoded as x_A; if the wire is forced to a lower-L route, this is encoded as $\overline{x_A}$.

Every net n is given a boolean variable. Therefore, $|U| = |S|$. The entire set of $\frac{n(n-1)}{2}$ interaction relations are encoded as specified. Each of these relations becomes a clause in the 2SAT instance.

Lemma 3.1:$|C| = O(|S|^2)$

Proof: $n = |S|$. Since there are at most two relations per interactions, $|C| \leq |S|(|S|-1)$. □

The 2SAT instance is obtained by letting each net n be a boolean variable $\in U$. The set of clauses C are the encoded net interactions.

Theorem 3.1: The coupling-free routing decision problem can be solved in $O(n^2)$ time where $n = |S|$

Proof: The CFRDP $\propto$ 2SAT in $O(n^2)$ time. An instance of 2SAT can be solved in linear time. Therefore, we can solve the coupling-free routing decision problem in $O(n^2)$ time.

4. IMPLICATION GRAPH

In this chapter, we show how an instance of the CFR problem is transformable into an *implication graph*. Then, we define some properties associated with the implication graph. We can utilize the properties of the implication graph to solve the CFR problem.

4.1 2SAT $\propto$ Implication Graph

First, we show how an instance of 2SAT is transformable into an implication graph. In Chapter 3, we show how to transform an instance of the CFR problem to an instance of 2SAT. Since CFR $\propto$ 2SAT $\propto$ implication graph, CFR $\propto$ implication graph. The multi-step transformation allows us to elegantly prove many properties associated with the implication graph. But, we will also show how to directly transform the CFR problem to an implication graph.

Let $\Lambda = \Lambda_i(x_i \vee y_i)$ be an instance of 2SAT, where x_i, y_i are literals over $a_1, ..., a_n \in A$. We want to know when SAT(A) is true. Define a digraph G = (V,E) by letting V be the set of literals and (x,y) $\in$ E if and only if $\overline{x} \vee y$ is one of the clauses. Recall that $\overline{x} \vee y$ is equivalent to $x \Rightarrow y$ (implication). We can assume there is no clause of the form $x \Rightarrow x$ since that is always true. Finally, note that $x \Rightarrow \cdots \Rightarrow y$ implies $x \Rightarrow y$.

Theorem 4.1: If there is a cycle in G containing both x and $\overline{x}$ for all x $\in$ V, A is not SAT.

Proof: The reason is that if x $\Rightarrow \overline{x}$, then x must be false. But since there is a cycle $\overline{x} \Rightarrow$ x which means x must be true. We have a contradiction. Therefore, A is SAT iff G does not contain any cycles including x and $\overline{x}$ for any literal x. $\square$

We call the digraph G an implication graph since it models the implications between the literals.

4.2 Coupling-Free Routing $\propto$ Implication Graph

Now we show how the CFR problem is directly transformable into an implication graph.

Given a set of nets N. The implication graph is a directed graph (digraph) $G(V, E)$. Let every vertex $v \in V$ correspond an upper-L routing and lower-L routing of each net $n \in R$. Therefore, $|V| = 2 \times |N|$. Then, $(x, y) \in E$ if and only if x forces y or, equivalently, $x \Rightarrow y$. We call this an implication.

Theorem 4.2: If there is an implication $x_A \Rightarrow x_B$, there is contrapositive implication $\overline{x_B} \Rightarrow \overline{x_A}$.

Proof: Since $x_A \Rightarrow x_B$, the upper-L routing of x_A must couple the lower-L routing of x_B. Therefore, a lower-L routing of net B ($\overline{x_B}$) will force a lower-L routing of net A ($\overline{x_A}$). $\square$

Theorem 4.3: Given a set of nets N, the construction of the corresponding implication graph takes running time $O(|N|^2)$.

Proof: First, we must determine the forcing interactions between every net. There are $\frac{|N|(|N|-1)}{2}$ possible interactions. Determining whether coupling exist in each interaction take $O(1)$ time. Therefore, it takes $O(|N|^2)$ time to determine the interactions. The number of vertices in the implication graph is exactly $2|N|$. The maximum number of edges is $\binom{2|N|}{2} = O(|N|^2)$. The forcing interactions determine whether or not an edge exist. This requires a simple $O(1)$ lookup into an interaction table. Adding up the complexities gives us the runtime of $O(|N|^2)$. $\square$

Lemma 4.1: Consider a set of nets N and its corresponding implication graph G. If there is a cycle in G containing x_i and $\overline{x_i}$ where i $\in$ N, then the nets N are not couple-free routable.

Proof: This is a direct consequence of Theorem 4.1. This should not be surprising since we can transform the CFR problem into 2SAT. $\Box$

Lemma 4.2: Given a set of nets N, there is an O($|V||E|$) algorithm to determine if these nets are coupling-free routable.

Proof: For sake of convenience, let $|N| = n$. Theorem 4.3 says that an implication graph is created in $O(n^2)$ time. Because of Lemma 4.1, if we find a cycle containing x_i and $\overline{x_i}$ the nets N are not 1-d routable. We can look for these cycles by doing a depth-first search from every vertex. If there is a path from x_i to $\overline{x_i}$ and a path from $\overline{x_i}$ to x_i, there is a cycle containing x_i and $\overline{x_i}$. We can do this for every vertex in O($|V||E|$). $O(n^2) < O(|V||E|)$. Therefore, we can determine if the nets are coupling-free routable in O($|V||E|$). $\Box$

For each implication case, up to two clauses are added to 2SAT in the transformation. These clauses correspond directly to edges in the implication digraph. Figure 4.1 shows a simple example for three nets. Focusing on nets A and B, we see that an upper-L routing of net A forces a lower-L routing for net B (corresponding to case 6). Therefore, we add the clause $(\overline{x_A} \vee \overline{x_B})$ to the 2SAT instance. In the implication graph, we add an edge from vertex x_A to vertex $\overline{x_B}$. Notice that an upper-L routing of net B forces a lower-L routing of net A. This corresponds to $x_B \Rightarrow \overline{x_A}$ which is the contrapositive of the previous statement. The other cases are similar. Notice that there are no cycles in the implication graph in Figure 4.1 (c). This means that these three nets can be coupling-free routed.

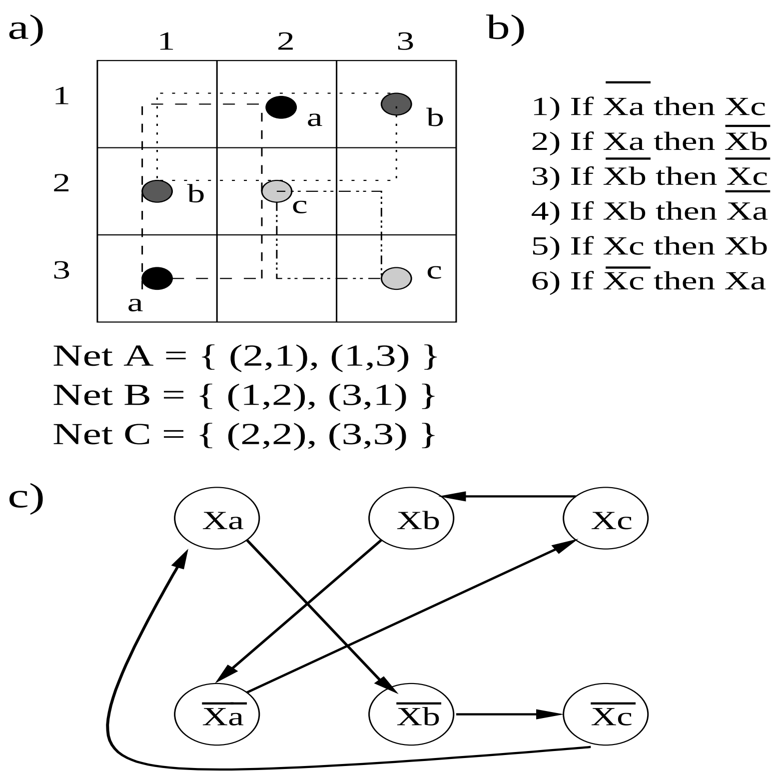

Figure 4.1: a) The layout of nets A, B and C. b) The implications of the nets. c) The implication graph. x_i indicates an upper-L routing of net i. The implication graph does not have any cycles containing x_i and $\overline{x_i}$, i $\in$ A,B,C, therefore the nets are coupling-free routable.

Given a set of boolean variables, U, a collection of clauses C such that each clause $c \in C$ has $|c| = 2$ and an integer $K \leq |U|$, the MAX2SAT problem is defined as finding a truth assignment for U such that at least K clauses $\in C$ are satisfied. MAX2SAT is NP-complete [40]. Therefore, finding the maximum subset of nets such that they can be coupling-free routed is equivalent to solving the MAX2SAT problem on the transformed 2SAT instance. This holds for any CFR problem with or without restrictions (e.g. routing must be planar). Note, this does not prove NP-completeness but gives you an idea about the complexity of the problem.

4.3 Properties

4.3.1 Direct Forcing

Assume that we have implication graph $G(V, E)$ which is constructed from an instance of a CFR problem containing the set of nets N. Remember that every vertex in the implication graph corresponds to a routing of a net $\in N$. Therefore, there are two vertices per net, one vertex for the upper-L routing and one vertex for the lower-L routing. We define the routing corresponding to vertex v as $route(v)$. Let $u, v \in V$ be two unique vertices. If there is a directed edge (u, v), then the route(u) forces route(v). This is a direct consequence of the way that the implication graph is constructed.

The *outdegree* of vertex v in a digraph is the number of vertices adjacent to v [14]. In an implication graph, the outdegree of v corresponds to the number of routings that route(v) forces. We call this a *direct forcing*.

4.3.2 Indirect Forcing

A routing may force a net even if it isn't a direct forcing. Referring to Figure 4.2, Route 1 directly forces only one route, Route 2. But, Route 2 forces Route 3 which forces Route 4. So, if we choose to route the net A in an upper-L manner (Route 1), then nets B, C and D must be laid out as Routes 2, 3 and 4, respectively if we want to route every net. Route 1 forces three routes even though it only directly forces Route 2. We say that Route 1 indirectly forces Routes 2, 3 and 4.

Given an implication graph $G(V, E)$ and vertices $u, v \in V$. A v *indirectly forces* u if there is a path from u to v. The number of total forcings (direct and indirect) of v is calculated by determining number of vertices that are connected to v. Algorithm 1 is a quick and simple method to determine the forcings of a route. It is a slightly modified version of depth-first search as described in [57].

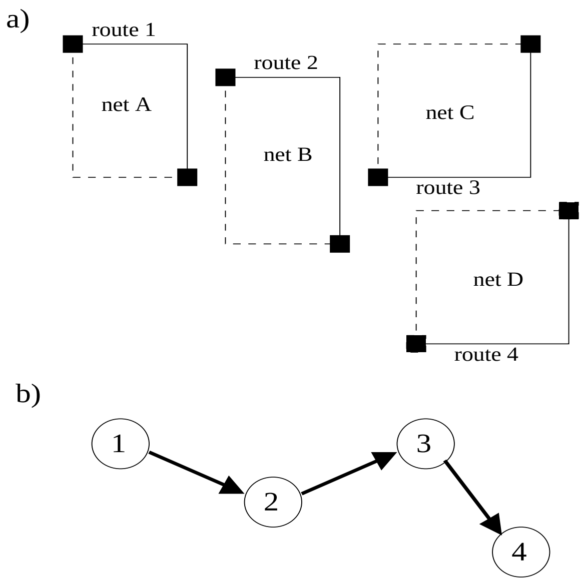

Figure 4.2: a) The layout of the nets b) The implication graph for routes 1,2,3 and 4.

Algorithm 1 *Forcings(A) – Returns the Total Number of Nets Forced by Route A*

Given an implication graph $G(V, E)$, a vertex u corresponding to Route A
forcings $\leftarrow$ 0
for each vertex $v \in V$
    do color[v] $\leftarrow$ white
DFS-Visit(u)
return forcings

Theorem 5.4: The running time of algorithm Forcings(A) is $O(|V| + |E|)$.

Proof: The "for" loop of the algorithm takes time $O(|V|)$. The procedure DFS-Visit is called at most once for each vertex $u \in V$ since DFS-Visit is invoked only on white vertices and the first thing it does is paint the vertex gray. During an execution of DFR-Visit(u), the loop is executed $|Adj[u]|$ times. Since

$$\sum_{u \in V} |Adj[u]| = O(E),$$

the total cost of the "for" loop in DFS-Visit is $O(E)$. Therefore, the time of running

Algorithm 2 *DFS-Visit(u)*

Assumes that forcings is global variable
color[u] $\leftarrow$ GRAY
forcings $\leftarrow$ forcings + 1
for each $v \in$ Adjacent[u]
    do if color[v] = WHITE
        then DFS-Visit(v)
color[u] $\leftarrow$ BLACK

Forcings(A) is $O(|V| + |E|)$. $\square$

5. MAXIMUM COUPLING-FREE LAYOUT

The Maximum Coupling-Free Layout Problem (MAX-CFL): Given a set of two-terminal nets S and a positive integer $K \leq |S|$. Is there a single-bend routing for at least K nets in S such that no two routings couple?

Theorem 5.1: The Maximum Coupling-Free Layout Problem for planar layouts is NP-Complete.

Proof: We make a transformation from the MAXWIRE problem. The MAXWIRE problem is defined as finding a set of nets T where $T \subseteq S$ and $|T| \geq K$ such that all the wires in T can be laid out in a single bend fashion on one layer. The MAXWIRE problem is NP-Complete [53]. By setting the coupling variables $d = \infty$ and $l = 0$, we can directly transform any instance of MAXWIRE to an instance of MAX-CFL. This essentially removes any coupling restrictions from the problem. □

MAX-CFL can be extended to consider criticality. The criticality of a net can be defined in numerous ways. Most often, a nets criticality is determined by the amount of timing slack that is available to that net. Also, the length of a net can be used. If we consider criticality, MAX-CFL tries to route a subset of nets with maximum criticality. A subset with maximum criticality will not always be the subset of maximum size.

Additional routing restrictions to the MAX-CFL problem are often needed. For example, we can use MAX-CFL to find a subset of planar nets. In this case, we must slightly modify the algorithms to consider intersection between the nets. Another common routing problem allows two layers to route the nets – one for vertical segments, one for horizontal segments. In this case, we must consider overlap between the nets. The algorithms that we present next assume that there are no restrictions. With the proper simple modifications, they can consider such restrictions.

Now, we look at a few heuristics to solve the MAX-CFL problem.

5.1 Greedy Algorithm

The first and most obvious algorithm that we consider is the greedy algorithm. This algorithm chooses the most critical net and, if possible, routes the net in an upper-L or lower-L fashion. If both the upper-L and lower-L routings couple with net that has already been laid out, the current net is not laid out; the most critical remaining net is then considered. The algorithm iterates until all nets have been considered.

Algorithm 3 *Maximum Coupling-Free Layout Routing Greedy Heuristic*

Given a set of nets N
Sort N by criticality (largest $\rightarrow$ smallest)
for each net $n \in N$
  do route n in upper-L or lower-L, if possible

Theorem 5.2: The Maximum Coupling-Free Routing Greedy Heuristic takes $O(n \log n)$ time.

Proof: The sorting step takes $O(n \log n)$ time. The "for" loop will complete after n iterations. Hence an $O(n \log n)$ run time for the algorithm. $\square$

The greedy heuristic is a simple and fast method of finding a maximum coupling-free layout solution.

Of course, there are many shortcomings to this algorithm. First, the greedy nature of the algorithm may cause a critical net that couples with many other less critical nets to be routed. By not routing a critical net, you may be able to route a large number of other less-critical nets which can lead to a better overall solution.

A simple example of this situation is shown in Figure 5.1. The greedy algorithm will place net A first. Then, it will place net B in an upper-L routing because it is the most critical unrouted net. Now, neither net C or net D can be placed since they both couple with net B. The best solution in terms of number of nets routed and total criticality routed is routing nets A, C and D.

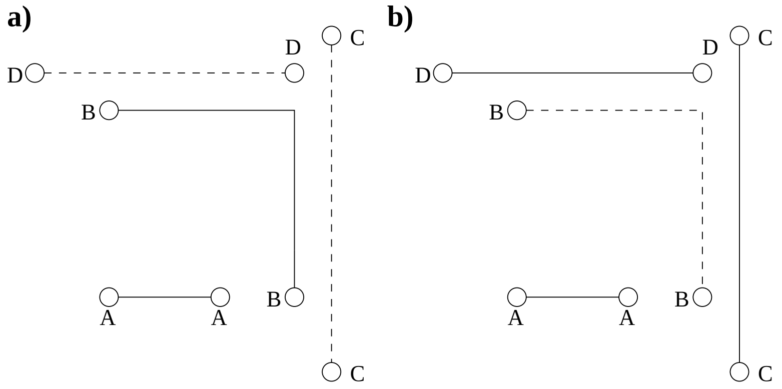

Figure 5.1: An unrouted net is displayed as a dotted line; a routed net has a solid line. Assume that criticality of net $A = 100$; $B = 50$; $C = D = 40$. a) Greedy algorithm solution. 2 nets are placed with a total criticality of 150. b) Best solution. 3 nets are placed with a total criticality of 180.

5.2 Direct Forcing Algorithm

The direct forcing algorithm tries to eliminate the bad decisions made by the greedy algorithm. It starts by determining the forcing interactions between every pair of nets. Then, it finds the nets that have a truly independent routing (either upper-L or lower-L) and routes them in the appropriate manner. An independent routing is equivalent to a route that forces no other nets (corresponding to interactions 1, 3-6 from Figure 3.1). If a net only forces other nets when it is routed in a lower-L (upper-L) will be routed in an upper-L (lower-L). The upper-L situations corresponds to interactions 3 and 5 while the lower-L situations corresponds to interactions 4 and 6. Since these routings are independent, routing these nets can not cause a situation as described in Figure 5.1. The remaining nets are routed according to the number of nets that they directly force. The net that directly forces the least amount of other nets is routed first, as long it doesn't couple with any net that is already routed. This process continues until all of the nets have been considered.

Algorithm 4 *Maximum Coupling-Free Layout Routing Forcing Heuristic*

Given a set of nets N
Determine the (direct) forcing interactions between the nets N
Route any independent nets
$N \leftarrow N-$ independent nets
$R \leftarrow \emptyset$
for each net $n \in N$
  do $R \leftarrow R \cup$ n.upper-L $\cup$ n.lower-L
Sort R by number of forcings (smallest $\rightarrow$ largest)
for each routing $r \in R$
  do if net associated with r is unrouted and r is routable
    then route r

Theorem 5.3: The Maximum Coupling-Free Routing Forcing Heuristic takes $O(n^2)$ time.

Proof: It takes $n(n-1)/2 = O(n^2)$ operations to determine the forcing interactions between the nets. The "for" loops both run in linear time. Sorting the set R takes $O(n \log n)$ times. This yields an overall runtime of $O(n^2)\Box$

5.3 Implication Algorithm

We showed how to generate an implication graph from an instance of the coupling-free routing problem in Chapter 4. Now, we use some of the properties of the implication graph to create a heuristic to solve the MAX-CFL problem.

First, the algorithm determines the number of direct and indirect forcings of each route. Then, it sorts the routes according a function of direct forcing and indirect forcings. Finally, it tries to route the nets based on this ordering.

The algorithm is similar in many ways to the direct forcing algorithm. In fact, this algorithm is exactly similar if you use a linear function ($direct\_forcings + \alpha \cdot indirect\_forcings$) and set $\alpha = 0$. The direct forcing algorithm does not use the implication graph simply because it would increase the runtime complexity. But, it could easily use the implication graph and determine the number of direct forcings based on the outdegree of each vertex.

Theorem 5.5: The running time of the implication algorithm is $O(|N|^3)$.

Proof: According to Theorem 4.3, the construction of the implication graph takes $O(|N|^2)$ time. There are $O(|N|)$ vertices in the implication graph, therefore the first "for" loop has $O(|N|)$ iterations. As stated in Theorem 5.4, the Forcings algorithm has a run time of $O(|V| + |E|)$. Note that $O(|V|) = O(|N|)$ and $O(|E|) = O(|N|^2)$. Therefore, the total run time of the "for" loop is $O(|N|^3)$. Sorting takes $O(|N| \log |N|)$ time. The final "for" loop $O(|N|)$ time. Therefore, the algorithm requires $O(|N|^3)$ time. □.

Algorithm 5 *Maximum Coupling-Free Layout Routing Implication Heuristic*

```
Given a set of nets N
Create an implication graph G(V, E)
R ← ∅
for each vertex v ∈ V
      do r.net ← route(v)
         r.num_forcing ← Forcings(route(v))
         R ← R ∪ r
Sort R by function(direct_forcings,indirect_forcing) (smallest → largest)
for each routing r ∈ R
      do if r.net is unrouted and r is routable
           then route r
```

5.4 Evaluation

To perform our experiments, we used five MCNC standard-cell benchmark circuits [35] and five benchmarks from the ISPD98 suite [6]. The characteristics of the circuits are shown in Table 5.1. The MCNC circuits were placed into using the Dragon global and detailed placement engine [45] which is comparable in quality to commercial version of Timberwolf [63]. The ISPD98 benchmarks are slightly modified [46] so that they could be placed by the Timberwolf placement engine.

The MCNC benchmarks are relatively small compared to today's designs and are questionable to use with the fixed-die placement engines [1]. Yet, they are the only publicly available placement benchmarks and are widely used in literature. The ISPD98 benchmarks are larger and comparable in size to current ASIC designs.

Data file	Num Cells	Num Nets	Num Pins
MCNC benchmarks			
prim1	833	1156	3303
prim2	3014	3671	12014
avqs	21584	30038	84081
biomed	6417	7052	22253
struct	1888	1920	5407
ISPD98 benchmarks			
ibm01	12036	13056	45815
ibm05	28146	29647	127509
ibm10	68685	75940	298311
ibm15	161187	186991	716206
ibm18	210341	202192	819969

Table 5.1: Benchmark circuit information

Our experiments focus on reducing the added delay caused by coupling. Long nets (in terms of wirelength) have the greatest opportunity for coupling and have the largest amount of interconnect delay. Therefore, we look at the longest nets from each of these circuits. We assume that these nets will be placed on a two preferred layers – one for horizontal routings and one for vertical routings.

We compare the greedy algorithm, the forcing algorithm and the implication algorithm in terms of number of nets routed and criticality of the nets that are routed. Net criticality is normally defined at the logic synthesis stage and is a function of the amount of slack available on a net. Unfortunately, the benchmarks do not include timing information. Hence, we need another measure of criticality. It has been shown that the delay for a wire of length l increases at the rate of $O(l^2)$ without wiresizing, $O(l\sqrt{l})$ with optimal wiresizing and linearly with proper buffer insertion [24]. We did experiments using linear (l), l-root-l ($l\sqrt{l}$), and quadratic (l^2) functions. The criticality function can easily be changed to incorporate some other function.

Table 5.2 shows the fraction of nets that are placed by the greedy, direct forcing and implication algorithms. In this experiment, we used the linear function indirect_forcing + $direct\_forcing$ for the implication algorithm.

We can see that the direct forcing algorithm consistently finds a routing for a larger percentage of nets. Surprisingly, the implication algorithm lays out a similar number of routings compared to the greedy algorithm. Figure 5.2 shows the average percentage of routes placed by the three heuristics. The number is the averaged over all the benchmarks. Over all the experiments that we ran, the direct forcing and implication algorithm routes, on average, 3.38% and 1.51% more nets than the greedy algorithm, respectively. It is a bit surprising that the implication algorithm performs poorly compared to the direct forcing algorithm. We will comment on this phenomenon later.

Algorithm	Avg. routes laid out
greedy	30.28%
direct forcing	35.30%
implication	31.79%

Table 5.2: Percentage of routes laid out. Averaged over all experiments.

If we only look at the criticality of the nets routed, we see that the greedy algorithm is better than the direct forcing and implication algorithms. Figures 5.3 and 5.4 confirm that the greedy algorithm outperforms the other algorithms using

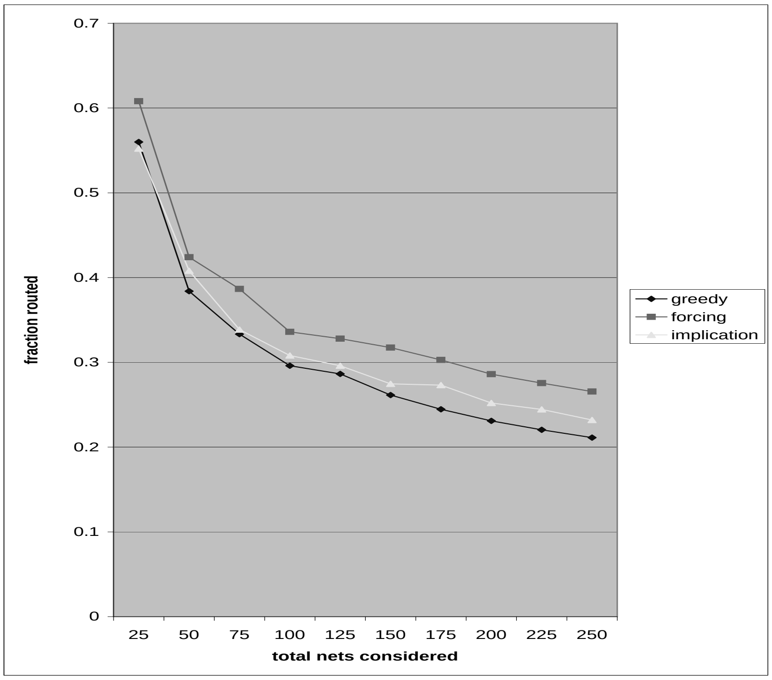

Figure 5.2: Fraction of nets placed averaged over all benchmarks.

a quadratic function, l-root-l and linear function. For a linear criticality function, the greedy algorithm was approximately a factor of 1.2 times better than the direct forcing algorithm. Using the same function, the greedy algorithm was approximately 1.1 times better than the implication algorithm. If we use the quadratic function, the greedy function outperforms the direct forcing and implication heuristics by a factor of 2.54 and 1.8, respectively (when we consider the 250 most critical nets). This should be of little surprise, however, since the direct forcing and implication algorithm do not use the idea criticality to find a routing of the nets.

It is somewhat surprising that the direct forcing algorithm leads to a layout with a larger number of nets than the implication algorithm. The direct forcing algorithm considers only direct forcings while the implication algorithm considers both direct and indirect forcings. Therefore, intuitively, the implication algorithm should perform better than the direct forcing algorithm. But, the results show otherwise. In fact, in terms of routes laid out, the implication algorithm barely

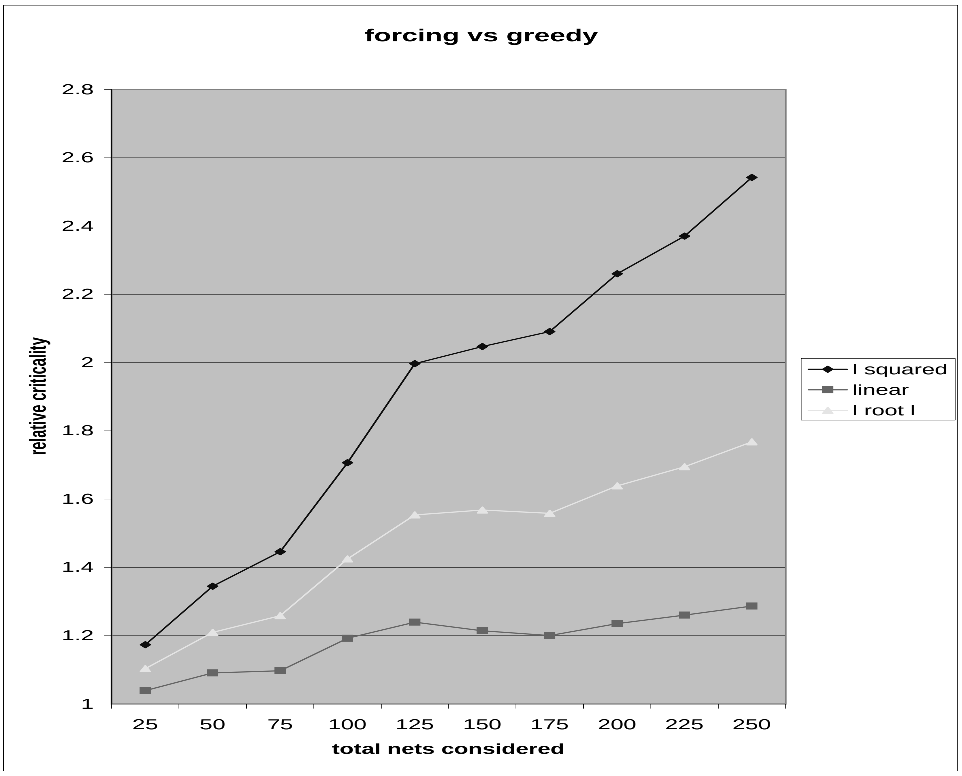

Figure 5.3: Relative criticality of nets placed by the greedy algorithm compared to the forcing algorithm. The results are averaged over all benchmarks. The criticality of the benchmarks are normalized to the criticality result of the forcing algorithm. Therefore, a result of x indicates that the greedy algorithm laid out $x\times$(criticality of forcing algorithm).

outperforms the greedy algorithm.

Through further investigation, we found some interesting data. First, we found that a majority of the nets have some sort of forcing interaction (direct or indirect) with every other net. That is, there is very little deviation in the total number of forcings over all of the nets. Since the implication algorithm sorts the nets on the total number of forcings and most of the nets have the same number of forcings, the implication algorithm is essentially random. This seems to explain the poor performance of the implication algorithm.

In light of those discoveries, we varied the function in the implication algorithm. From this point forward, the notation "implication(α)" denotes that the

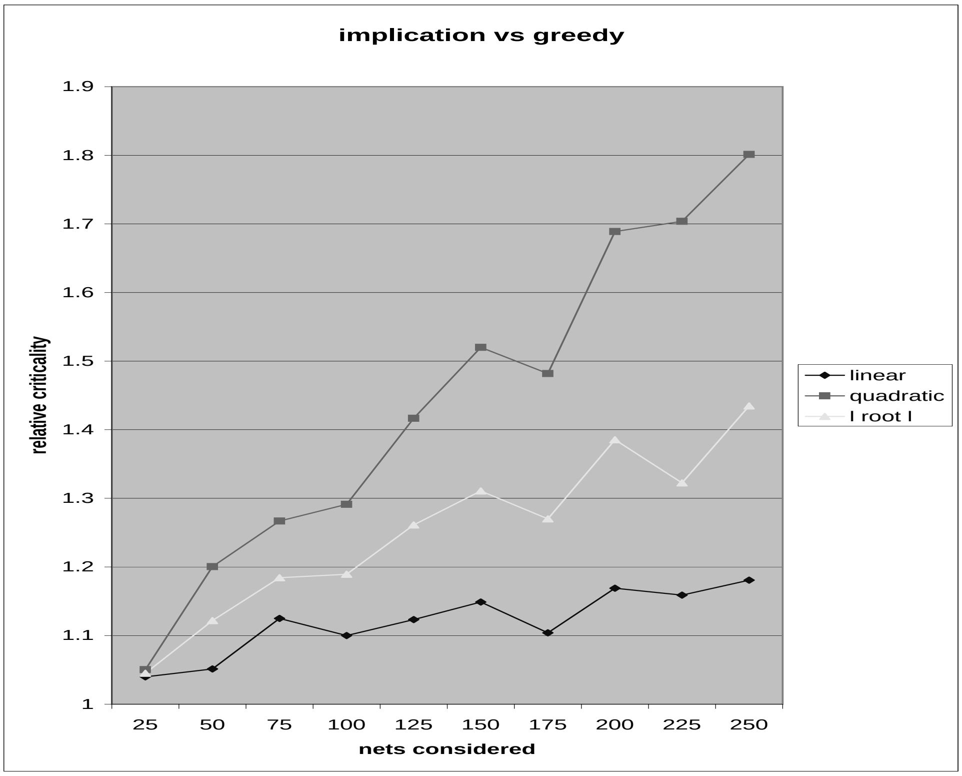

Figure 5.4: Relative criticality of nets placed by the greedy algorithm compared to the implication algorithm. The results are averaged over all benchmarks. The criticality of the benchmarks are normalized to the criticality result of the implication algorithm. Therefore, a result of x indicates that the greedy algorithm laid out $x\times$(criticality of implication algorithm).

implication algorithm used the function $indirect\_forcing + \alpha \cdot \, direct\_forcing$. We focused the investigation on linear functions. If $\alpha > 1$, the algorithm focuses on laying out nets with minimum number of direct forcing and the number of indirect forcings acts as secondary value. If $0 \leq \alpha < 1$, the algorithm emphasizes the number of indirect forcings; the direct forcings are secondary in this case.

Table 5.3 shows the overall percentage of routed nets for various values of α. Table 5.4 shows the relative criticality of the various implication algorithms compared to the direct forcing algorithm. The factor α seems to help the implication algorithm find a better solution. In fact, the percentage of routes that are laid out increases by 3.66% by varying α from 2 to 10 as compared to the original implication algorithm ($\alpha = 1$). In terms of percentage of laid out nets, the implication algorithm slightly outperforms the direct forcing algorithm. Comparing Table 5.2 and 5.3, we can see that the implication algorithm routes the highest percentage of nets over all the algorithms. The criticality of the nets placed also increases when $\alpha = .5$, 2, 5 and 10, though it is still inferior in all cases when compared to the criticality results of the greedy algorithm.

Algorithm	Percentage routes laid out
direct forcing	35.30%
implication(1)	31.79%
implication(2)	35.45%
implication(5)	35.45%
implication(10)	35.45%
implication(.5)	35.45%
implication(∞)	35.30%

Table 5.3: Percentage of routes laid out while varying the implication function. Averaged over all experiments.

Algorithm	Rel. l crit	Rel. $1\sqrt{l}$ crit	Rel. l^2 crit
direct forcing	1.000	1.000	1.000
implication(1)	1.062	1.169	1.296
implication(2)	1.078	1.116	1.156
implication(5)	1.078	1.116	1.156
implication(10)	1.078	1.116	1.156
implication(.5)	1.078	1.116	1.156
implication(∞)	1.000	1.000	1.000

Table 5.4: Criticality of the implication algorithm (using various linear functions) relative to the direct forcing algorithm. The fraction is $\frac{implication}{directforcing}$. Averaged over all experiments.

The results show that the implication(∞) algorithm is exactly similar to the direct forcing algorithm in terms of criticality and percentage routes laid out. This should not be surprising since, as we stated earlier, these two algorithms are identical. Also, it seems that when $\alpha = .5$, 2, 5 and 10, the algorithms produce the same layout. This was not formally verified, but the result in terms of nets laid out and criticality are exactly the same for every experiment. It is highly unlikely that this is coincidental.

In summary, the results indicate that the implication algorithm is the best algorithm for routing the maximum number of nets. The greedy algorithm tends to find a layout with maximum criticality but performs poorly with respect to maximizing the number of nets.

6. CONCLUSION

In this work, we present the Coupling-Free Routing (CFR) Problem. It addresses the issue of coupling in routing. The CFR problem considers only one-bend routings which guarantees that each net is routed with minimum wirelength, minimum number of vias and minimum delay. Additionally, one-bend routes limit the potential number of routings to two, making it easier to predict metrics such as congestion and wirelength earlier in the design flow. We purposely define CFR problem to be somewhat generic. This allows us to use the problem in a wider variety of algorithms. We argue that these problems are useful in detailed routing since they:

1. can insure that critical nets are routed with minimal coupling on a preferred layer.

2. can be incorporated into existing single layer routings algorithms, making these algorithms more useful as we go further into the DSM era.

Additionally, these problems are useful in global routing to help guide the global router to a coupling-free detailed layout.

We introduce a new type of graph, the implication graph, which takes an instance of the CFR problem and models the dependencies or forcings that exists between the nets.

We present an exact, efficient algorithm for the CFR decision problem via a transformation to the 2-satisfiability problem. The CFR decision problem will determine whether every net within a specified set is coupling-free routable.

The MAX-CFL problem is defined as finding a coupling-free routing for the maximum number of nets in a set. We show that the planar MAX-CFL problem is NP-Complete. Also, we give three heuristics for solving the general MAX-CFL problem, the greedy, direct forcing and implication algorithms. The greedy algorithm is quite simple, yet is an effective way of obtaining a layout with maximal criticality. Of the three algorithms, the greedy algorithm has the smallest runtime complexity. The direct forcing algorithm uses the notion of direct forcing to find a MAX-CFL solution. In terms of routing the maximum number of nets, this algorithm performs around 3% better than the greedy algorithm. The final algorithm we developed is the implication algorithm. The implication algorithm uses some properties associated with the implication graph. Our experiments show that the implication algorithm is the best algorithm at routing the maximum number of nets; it consistently routes the largest number of nets.

In the future, we plan on developing better algorithms for solving the MAX-CFL problem. Specifically, we hope to incorporate criticality data into the forcing and implication algorithm to improve its performance with respect to criticality. Also, we plan on incorporating these algorithms into our existing global and detailed routers.

[1] A. Caldwell, A. Kahng and I. Markov. "Can Recursive Bisection Alone Produce Routable Placements?". In *Proc. ACM/IEEE Design Automation Conference* (June 2000).

[2] A. Devgan. "Efficient Coupled Noise Estimation for On-Chip Interconnects". In *Proc. IEEE International Conference on Computer Aided Design* (November 1997).

[3] A. Kahng and G. Robins. "A New Class of Steiner Tree Heuristics with Good Performance: The Iterated 1-Steiner Approach". In *Proc. IEEE International Conference on Computer Aided Design* (November 1990).

[4] A. Kahng, S. Mantik and D. Stroobandt. "Requirements for Models of Achievable Routing". In *Proc. International Symposium on Physical Design* (April 2000).

[5] A.V. Aho, M.R. Garey and F.K. Hwang. "Rectilinear Steiner Trees: Efficient Special-Case Algorithm". *Networks* (July 1977).

[6] C. Alpert. "The ISPD98 Circuit Benchmark Suite". In *Proc. International Symposium on Physical Design* (April 1998).

[7] C.Y. Lee. "An Algorithm for Path Connection and Its Application". *IRE Transactions on Electronic Computer* (1961).

[8] D. Kirkpatrick and A. Sangiovanni-Vincentelli. "Techniques for Crosstalk Avoidance in the Physical Design of High-Performance Digital Systems". In *Proc. International Conference on Computer Aided Design* (November 1994).

[9] D. Richards. "Fast Heuristic Algorithms for Rectilinear Steiner Trees". *Algorithmica* (April 1989).

[10] D. SYLVESTER AND K. KEUTZER. "A Global Wiring Paradigm for Deep Submicron Design". *IEEE Transactions on Computer Aided Design* (February 2000).

[11] D. SYLVESTER *et al.* "Interconnect Scaling: Signal Integrity and Performance in Future High-speed CMOS Designs". In *Proc. of VLSI Symposium on Technology* (1998).

[12] E.F. MOORE. *"The Shortest Path through a Maze"*. Annals of the Harvard Computation Laboratory, Vol. 30, Pt. II, 1959.

[13] F. DARTU AND L. PILEGGI. "Calculating Worst-case Gate Delays Due to Dominant Capacitance Coupling". In *Proc. ACM/IEEE Design Automation Conference* (June 1997).

[14] G. CHARTRAND AND O. OELLERMANN. *Applied and Algorithmic Graph Theory.* McGraw-Hill, Inc., 1993.

[15] G. DE MICHELI. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, Inc., 1994.

[16] G. VIJAYAN, N. HASAN AND C.K. WONG. "A Neighborhood Improvement Algorithm for Rectilinear Steiner Trees". In *IEEE International Symposium on Circuits and Systems* (May 1990).

[17] G. YEE *et al.* "Wire Delay in the Presence of Crosstalk". In *Proc. of TAU* (1997).

[18] GAREY, M., AND JOHNSON, D. "The Rectilinear Steiner Tree Problem is NP-Complete". *SIAM Journal on Applied Mathematics* (March 1977).

[19] H. Chang et al. "A Top-down, Constraint-driven Design Methodology for Analog Integrated Circuits". In *Proc. IEEE Custom Integrated Circuits Conference* (1992).

[20] H. Zhou and D.F. Wong. "An Optimal Algorithm for River Routing with Crosstalk Constraints". In *Proc. IEEE International Conference on Computer Aided Design* (November 1996).

[21] H. Zhou and D.F. Wong. "Global Routing with Crosstalk Constraints". In *Proc. ACM/IEEE Design Automation Conference* (June 1998).

[22] H.B. Bakoglu. *Circuits, Interconnections and Packaging for VLSI.* Addison-Wesley Publishing Company Inc., 1990.

[23] H.P. Tseng, L. Scheffer and C. Sechen. "Timing and Crosstalk Driven Area Routing". In *Proc. ACM/IEEE Design Automation Conference* (June 1998).

[24] J. Cong and D.Z. Pan. "Interconnect Delay Estimation Models for Synthesis and Design Planning". In *Proc. Asia and South Pacific Design Automation Conference* (January 1999).

[25] J. Cong and P. Madden. "Performance Driven Multi-Layer General Area Routing for PCB/MCM Designs". In *Proc. ACM/IEEE Design Automation Conference* (June 1998).

[26] J. Cong *et al.* "Performance Optimization of VLSI Interconnect Layout". *Integration, the VLSI Journal* (1996).

[27] J. Cong *et al.* "Interconnect Design for Deep Submicron ICs". In *Proc. IEEE International Conference on Computer Aided Design* (November 1997).

[28] J. Cong, J. Fang and K.-Y. Khoo. "DUNE: A Multi-Layer Gridless Routing System with Wire Planning". In *Proc. International Symposium on Physical Design* (April 2000).

[29] J. Ho, G. Vijayan and C.K. Wong. "A New Approach to the Rectilinear Steiner Tree Problem". In *Proc. ACM/IEEE Design Automation Conference* (June 1989).

[30] J. Rabaey. *Digital Integrated Circuits: A Design Perspective.* Prentice-Hall, Inc., 1996.

[31] J.M. Ho, G. Vijayan and C.K. Wong. "New Algorithm for the Rectilinear Steiner Tree Problem". *IEEE Transactions on Computer Aided Design* (February 1990).

[32] J.M. Smith, D.T. Lee and J.S. Liebman. "An O(n log n) Heuristic Algorithm for the Rectilinear Steiner Minimal Tree Problem". *Engineering Optimizations* (April 1980).

[33] K. Chaudhary, A. Onozawa and E.S. Kuh. "Cross Point Assignment with Global Rerouting for General-Architecture Designs". In *Proc. IEEE International Conference on Computer Aided Design* (November 1993).

[34] K. Jhang, S. Ha and C.S. Jhon. "COP: A Crosstalk Optimization for Gridded Channel Routing". *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems* (April 1996).

[35] K. Kozminski. "Benchmarks for Layout Synthesis - Evolution and Current Status". In *Proc. ACM/IEEE Design Automation Conference* (June 1991).

[36] K.F. Liao, M. Sarrafzadeh and C.K. Wong. "Single-Layer Global Routing". *IEEE Transactions on Computer Aided Design* (1994).

[37] K.L. Shepard. "Design Methodologies for Noise in Digital Integrated Circuits". In *Proc. ACM/IEEE Design Automation Conference* (June 1998).

[38] K.L. Shepard and V. Narayanan. "Conquering Noise in Deep-Submicron Digital ICs". *IEEE Design and Test of Computers* (January-March 1998).

[39] K.W. Lee. *"Global Routing of Row-Based Integrated Circuits"*. PhD thesis, Yale University, May 1977.

[40] M. Garey and D. Johnson. *"Computers and Intractability: A Guide to the Theory of NP-Completeness"*. W.H. Freeman and Company, New York, NY, 1979.

[41] M. Guruswamy and D.F. Wong. "A General Multi-layer Area Router". In *Proc. ACM/IEEE Design Automation Conference* (June 1991).

[42] M. Lee, A. Hill and M.H. Darley. "Interconnect Inductance Effects on Delay and Crosstalk for Long On-Chip Nets with Fast Input Slew Rates". In *Proc. International Symposium on Circuits and Systems* (May 1998).

[43] M. Sarrafzadeh and C.K. Wong. "Hierarchical Steiner Tree Construction in Uniform Orientations". *IEEE Transactions on Computer Aided Design* (August 1992).

[44] M. Sarrafzadeh and C.K. Wong. *An Introduction to VLSI Physical Design.* McGraw-Hill, New York, NY, 1996.

[45] M. Sarrafzadeh and M. Wang. "NRG: Global and Detailed Placement". In *Proc. IEEE International Conference on Computer Aided Design* (November 1997).

[46] M. Wang, X. Yang and M. Sarrafzadeh. "DRAGON: Fast Standard-Cell Placement for Large Circuits". In *Proc. IEEE International Conference on Computer Aided Design* (November 2000).

[47] N. Sherwani. *"Algorithms For VLSI Physical Design Automation"*. Kluwer Academic Publishers, Boston, MA, 1993.

[48] N.K. Sehgal et al. "A Gridless Multi-layer Area Router". In *Proc. Great Lakes Symposium on VLSI* (March 1991).

[49] P. Hart, N. Nilsson and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Coast Paths". *IEEE Transactions on Systems, Science and Cybernetics* (1968).

[50] R. Dutta et al. "Multi-layer Area Routing Algorithms and Optimization Problem". In *Proc. IEEE Custom Integrated Circuits Conference* (1997).

[51] R. Kastner, E. Borzorgzadeh and M. Sarrafzadeh. "Predictable Routing". In *Proc. IEEE International Conference on Computer Aided Design* (November 2000).

[52] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh. "Coupling Aware Routing". In *Proc. IEEE International ASIC/SOC Conference* (September 2000).

[53] R. Raghavan, J. Cohoon and S. Sahni. "Single Bend Wiring". *Journal of Algorithms* (June 1986).

[54] R.M. Karp. *"Reducibility Among Combinatorial Problems"*. Complexity of Computer Computations, New Rouk: Plenum, 1972.

[55] S. Even, A. Itai and A. Shamir. "On the Complexity of Timetable and Multicommodity Flow Problems". *SIAM Journal of Comp.* (1976).

[56] Semiconductor Industry Association. *"National Technology Roadmap for Semiconductors"*, 1997.

[57] T. Cormen, C. Leiserson and R. Rivest. *Introduction to Algorithms.* The MIT Press, McGraw-Hill, Inc., 1990.

[58] T. Gao and C.L. Liu. "Minimum Crosstalk Switchbox Routing". In *Proc. IEEE Internation Conference on Computer Aided Design* (November 1994).

[59] T. Lengauer. *"Combinatorial Algorithms for Integrated Circuit Layout".* John Wiley and Sons, New York, 1990.

[60] T. Xue, E.S. Kuh and D. Wang. "Post Global Routing Crosstalk Risk Estimation and Reduction". In *Proc. IEEE International Conference on Computer Aided Design* (November 1996).

[61] V. Vaishnavi and D. Wood. "Rectilinear Line Segment Intersection, Layered Segment Trees and Dynamization". *Journal of Algorithms* (July 1982).

[62] W. Dougherty and D. Thomas. "Unifying Behavioral Synthesis and Physical Design". In *Proc. ACM/IEEE Design Automation Conference* (June 2000).

[63] W.J. Sun and C. Sechen. "Efficient and Effective Placement for Very Large Circuits". In *Proc. International Conference on Computer Aided Design* (November 1993).

[64] Y. Massoud *et al.* "Layout Techniques for Minimizing On-Chip Interconnect Self Inductance". In *Proc. ACM/IEEE Design Automation Conference* (June 1998).

[65] Z.-M. Lin and Z.-W. Ro. "A Heuristic Planar Routing Algorithm for High Performance Single-Layer Layout". Manuscript, 2000.