Simultaneous Shield and Buffer Insertion for Crosstalk Noise Reduction in Global Routing

Tianpei Zhang and Sachin S. Sapatnekar
Department of Electrical and Computer Engineering, University of Minnesota
E-mail: zhangt, sachin@ece.umn.edu

***Abstract*— As VLSI technologies scale down, interconnect performance is greatly affected by crosstalk noise due to the decreasing wire separation and increased wire aspect ratio, and crosstalk has become a major bottleneck for design closure. The effectiveness of traditional buffering and spacing techniques for noise reduction is constrained by the limited available resources on chip. In this paper, we present a method for incorporating crosstalk reduction criteria into global routing under a broad power supply network paradigm. This method utilizes power/ground wires as shields between signal wires to reduce capacitive coupling, while considering the constraints imposed by limited routing and buffering resources. An iterative procedure is employed to route signal wires, assign supply shields, and insert buffers so that both buffer/routing capacity and signal integrity goals are met. In each iteration, shield assignment and buffer insertion are considered simultaneously via a dynamic programming-like approach. Our noise calculations are based on Devgan's metric, and our work demonstrates, for the first time, that this metric shows good fidelity on average. An effective noise margin inflation technique is also proposed to compensate for the pessimism of Devgan's metric. Experimental results on testcases with up to about 10,000 nets point towards an asymptotic runtime that increases linearly with the number of nets. Our algorithm achieves noise reduction improvements of up to 53% and 28%, respectively, compared to methods considering only buffer insertion, or only shield insertion after buffer planning.**

I. Introduction

With increasing operating frequencies and decreasing minimum feature size in nanometer VLSI design, interconnect performance issues have become dominant in determining the performance of a circuit. Besides optimizing interconnect timing and power for performance improvement, it is also important to integrate the analysis and optimization of interconnect crosstalk noise in order to maintain signal integrity. With each successive technology generation, wires are spaced closer to each other and also have more skewed aspect ratios, resulting in increased levels of coupling capacitance in each generation, which can cause a switching net to induce large noise spikes on its neighboring nets. Crosstalk noise can affect the circuit performance in one of two ways:

(a) *Functional noise* is seen when a victim net changes its level due to the switching of its neighbor aggressor nets, and this could lead to circuit malfunction.

(b) *Delay noise* is caused when the victim and aggressor nets switch at the same time, causing the injection of a noise pulse during switching, which can alter the delay.

The algorithm proposed in this paper primarily mitigates functional noise; however, the insertion of buffer and supply shields with stable voltage levels between signal wires also provides the subsidiary benefit of greatly easing delay uncertainties and therefore relieves the delay noise.

Various noise analysis techniques have been proposed over the years. Crosstalk noise can be most accurately evaluated by circuit simulation methods such as SPICE [22], but this is computationally expensive for large circuits. On the other end of the spectrum, most simplified, some existing noise optimizations in physical design [16], [34], [36] employ a geometric noise metric, in which the noise level is proportional to the overlap length between wires. However, such a model is not accurate since it does not capture the electrical properties of circuit. Under a linear circuit model, noise analysis can be performed using model order reduction techniques as in [23], through which noise waveforms can be obtained with high accuracy, although the computational cost can still be too high for its use in a full chip noise optimization tool. To further expedite noise analysis, various simplified crosstalk models have been developed. The work in [30] derives bounds for crosstalk noise using a lumped model, but ignoring interconnect resistance and assuming a step input for aggressor driver. The peak noise expression in [30] is extended by [31] to include a π circuit model for interconnect and a saturated ramp input model. This effort is further extended to a $2-\pi$ interconnect modeling in [8] to take into account the coupling location on the victim net and to better model the aggressor net. Although these approaches can provide closed-form analytical model for noise attributes, their simplified coupling model fails to capture the distributed nature of coupling between adjacent lines; at the same time, the closed-form expressions are too complicated to be incorporated in a optimization engine in physical design.

Devgan's work in [9] proposes a clever metric to estimate the peak noise, in which various circuit electrical properties, such as distributed coupling capacitance, ag-

gressor slew rate and wire resistance, are taken into consideration. Although pessimistic, its simple form make it amenable to be incorporated in physical design [1], [5], [19] due to the fact that it is easy to compute incrementally. This is particularly useful for methods that incrementally expand a partial route for a net until the complete route is determined.

As an extension to Devgan's noise metric, [18] employs moment-matching techniques to accurately estimate noise in a RLC network. However, this method requires multiple tree traversals to obtain an noise estimation, hence not easily embedded into the noise optimization process. Another extension to Devgan's metric is shown in [14], which introduces a time constant to control the exponential noise decay at victim node to reduce pessimism. This approach bears a simple form, and shows high accuracy for noise analysis in RC circuits. Nevertheless, compared with Devgan's metric, this method requires coupling information over the whole net to be available to obtain a valid noise solution, while Devgan's metric has the property of incrementality, and can generate a partial noise solution with incomplete coupling information during building noise protection solutions. Therefore, Devgan's metric can greatly reduce protection algorithm complexity, as will be detailed later. Moreover, the similarity of its computations to Elmore's delay metric [11], which many engineers in industry are very familiar with, imply that there is a low learning curve, and that intuitions from Elmore's metric can be carried over to this problem.

Based on the above review and comparison, we employ Devgan's noise metric, among other noise analysis models, in our noise reduction algorithm during global routing. This is mainly due to the fact that Devgan's metric considers a wide range of electrical properties and bears a extremely concise form as will be discussed in Section II-C.2. Furthermore, this paper studies the fidelity of Devgan's noise metric under our experimental setup and proposes a noise margin inflation technique to overcome its pessimism at high slew rates [14], [18] so as to to safely employ this metric. Therefore, Devgan's metric fits well into our global routing flow for noise reduction.

The efforts for crosstalk noise mitigation are exerted throughout physical design stages. In early design stages, [5] addresses the problem of buffer planning for better noise reduction during floorplanning. Due to the lack of detailed physical information at this early stage, the optimization efforts can be quite limited. On the other hand, after the routing phase, with more detailed physical and electrical properties of a circuit known, crosstalk noise can be effectively reduced with various techniques. Many recent works [4], [12], [27] present gate sizing methods to reduce noise by appropriately increasing the driving gate size of victim net and decreasing that of aggressor net. Wire spacing, which can control the separation between nets, are utilized to optimize crosstalk noise in [19], [20]. Similarly, wire sizing is employed in [13] for noise reduction. A disadvantage of the above techniques is that they are all performed after the routing phase. As the routes and gate positions are rather fixed, the solution space can be constrained, hence greatly limits the optimization efforts.

Among other physical design stages, routing determines the routes, layers and relative positions of nets, which are some of the most contributing factors to affect the noise level of a net; at the same time, some effective noise mitigation resources such as buffers and shields, can also be employed during routing. Therefore, noise reduction efforts *during* the routing phase can significantly improve the signal integrity of a VLSI circuit. Previous routing techniques [16], [17], [34], [36] use simple crosstalk metrics and/or ignore issues related to routing congestion, particularly due to supply nets and shields. A practical crosstalk-conscious router must consider the trade-off between routing resource consumption and noise reduction. To address this issue, shield planning and crosstalk budgeting are utilized in some recent works [15], [32], [33], [35] to perform global routing with RLC noise avoidance and routing area optimization under resource constraint. However, buffer insertion, an effective way to block noise propagation, is not addressed in these works. [1] proposes an algorithm to utilize buffers for improved timing and noise performance, but this technique is performed in the post-layout phase, therefore suffers from restrained solution space. Moreover, it does not fully address the contention for buffer resources. With trends showing the use of an increasingly large number of buffers to reduce interconnect delay and achieve timing closure in modern designs, it is projected at 32nm technology, a very large proportion of all cells need to be buffers [25]. This will produce high levels of contention for limited buffer resources, implying that buffer considerations must be taken into account during global routing, for best resource utilization. This motivates our simultaneous buffer and shield insertion scheme for functional noise reduction. Together, shielding wires and buffers can effectively control crosstalk noise, and an integrated approach can manage both resources optimally.

This paper addresses the problem of crosstalk noise reduction during global routing under restrictions on the availability of routing and buffer resources. Some of the contributions in this paper are as follows:

- We simultaneously allocate power supply wires as shields and insert buffers in global routing phase in order to route nets under a noise budget, so as to best utilize noise avoidance resource.
- We develop a practical methodology to tackle the pessimism of Devgan's metric. For the first time, we verify the fidelity of this metric, and develop a noise margin inflation technique to compensate for the over-estimation of Devgan's metric.
- To utilize existing power supply wires, this method is presented under the backdrop of a flexible supply network architecture. The procedure results in a

signal/power co-routing solution at the global level.
- We also incorporate considerations to insert a sufficient number of buffers to control the delays and slews on each signal line.
- Experiments show our algorithm achieves noise reduction improvements of up to 53% and 28%, respectively, compared to methods considering only buffer insertion, or only shield insertion after buffer planning; while testcases with up to about 10,000 nets point towards an asymptotic runtime that increases linearly with the number of nets.

Our method works iteratively: starting with an initial global routing solution, an enumerative dynamic programming-like algorithm is used to simultaneously assign supply shields and buffers to meet the noise budget for each net, one at a time, to find a minimum cost solution for the net. Next, an iterative rip-up-and-reroute step is performed to better meet the routing and noise goal. We simultaneously take into account the limitations on routing/buffer resources and the needs for signal integrity and provide a global routing solution that is immune to capacitive coupling noise. For comparison purposes, we also implement an intelligent greedy approach which is faster, but less effective in resource allocation. The organization of the paper is as follows. In Section II, the model and problem formulation are formally presented. This is followed by the verification of the validity of using Devgan's metric in Section III. Next, in Section IV, we describe our iterative algorithm. Experimental results are presented in Section V, followed by a conclusion in Section VI.

II. Preliminaries

A. Global Routing and Buffer Model

As shown in Figure 1, our global routing model tessellates the entire chip into an array of grid cells, referred to as the *routing grid*. *Net N* consists of a set of electrically equivalent pins $\{s, p_1, p_2, ..., p_k\}$ distributed in different routing grid cells, that must be connected by wires, of which s is the source and $p_1, p_2, ..., p_k$ are the sinks. The dual graph of the routing grid tessellation is the *routing graph* G, which is shown in Figure 1(a) as the dashed lines. Connections among all of the pins will be routed over the routing graph G. Each edge in the routing graph corresponds to a boundary e_{ij} in the routing grid that connects grid cells i and j. The *grid length*, L_e, is defined as the center-to-center distance between two neighboring grid cells. Due to geometrical limitations on the boundary, we require that $W_e \leq C_e$, in which W_e is the total width (including wire spacing) used by signal and power lines passing the boundary, and C_e is the geometrical width of the boundary e, or the *boundary capacity*. A violation of this requirement results in *boundary overflow*. We follow a routing model in which horizontal and vertical lines are routed on different layers. The aim of routing is to eliminate boundary overflows while achieving other performance-related goals.

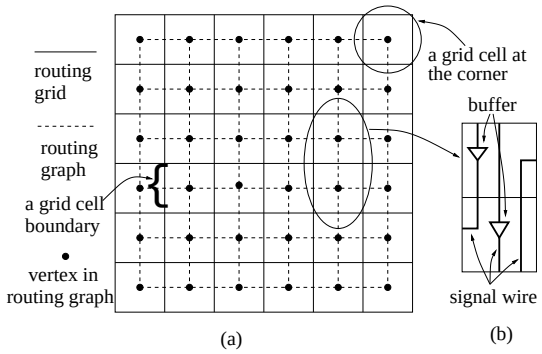

Fig. 1. Routing grid and buffer insertion for signal wires.

In the literature, various models for buffer placement have been proposed, prominent among which are buffer block planning [7], in which buffer blocks are placed *between* the building blocks of the circuit, and distributed buffer model [2], in which buffers are interspersed *within* the routing grid, and their exact location is undetermined until later in the design process. The latter approach has several advantages over the former in terms of reduced congestion and increased flexibility [2]. Therefore we adopt this distributed buffer model in our work, and specify it in more detail as follows. Figure 1(b) shows an inset view of a part of the routing grid where distributed buffers are inserted into a signal wire. For a grid cell i, the number of available buffers is denoted as B_i. If the number of utilized buffers is b_i, then $b_i \leq B_i$ must be satisfied, otherwise we have *buffer overflow*. To control the interconnect delay and slew rate, a maximum wire length between two buffers (gates) is enforced. For instance, in [2], [10], for a high-end microprocessor, consecutive buffers are separated by at most 4500μm. In terms of a constraint in our global routing model, this translates to a requirement that the maximum total interconnect length that can be driven by a buffer (gate) is the length of M grids.

B. Power Supply Architecture

A traditional power supply architecture is composed of a regular dense grid that traverses the entire layout area. However, different parts of the layout require various amounts of current, and the density of the grid does not have to be uniform. This flexibility is exploited in [26] to design a locally uniform, globally non-uniform power grid structure, and this property is also employed in [28] to optimize routability while achieving satisfactory power integrity.

In this work, we utilize a similar non-uniform and flexible power grid architecture that can satisfy the power requirements and provide shielding functionality between neighboring signal wires. We assume that the power grid is an array of variable density, and the power integrity of the supply grid is maintained by ensuring that the average and minimum number of wires feeding every block exceeds a specified threshold. The layout

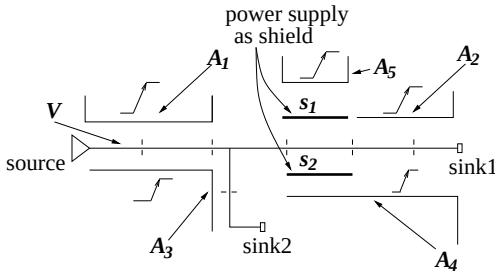

Fig. 2. Switching noise on a victim net with shielding supply wires. V is the victim net, A_i, $i = 1, \ldots, 5$ are potential aggressor nets, and s_1, s_2 are power supply shields.

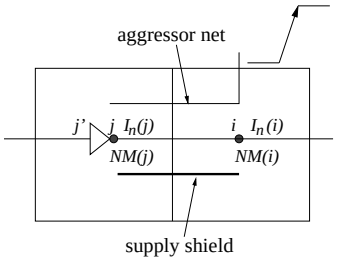

Fig. 3. Calculation of noise margin by Devgan's metric.

is divided into blocks, which corresponds to different functional blocks in practice. For each block i, regarding the power integrity, we are given:

- a Minimum Average Number (MAN) of supply wires MAN_i per grid edge.
- a Minimum Number (MN) of supply wires MN_i over each grid edge belonging to the block i.

Note that MAN_i and MN_i are both defined *per edge* of the routing graph of Section II-A and together define a basic structure of power network, thus enabling the power considerations to be incorporated even before a detailed power architecture is determined. Any extra power lines/shields beyond MN_i and MAN_i will only improve the power grid performance [28]. This power grid model works well on intermediate metal layers, as in [24], where the variable number of power lines in adjacent blocks do not have to match exactly since they can be connected to each other through upper layers. The edge capacities in the routing graph are shared by signal wires and power supply wires, implying that if signal wires utilize too much of the routing capacity at a boundary, then it is not possible to make enough room for supply wires. The supply wires in the supply grid are used not only to carry power currents, but also work as shields between aggressor and victim signal wires to reduce noise; we will use the terms *supply wire* and *shield* interchangeably.

C. Noise Calculation under Shield Insertion

1) Supply Shield Arrangement: The insertion of a supply wire between two signal wires will shield them from each other. As supply wire has a stable voltage level, it can effectively mitigate the capacitive coupling noise between the signal lines. We say that a side of a signal wire is provided *protection* if it neighbors a supply shield. Figure 2 shows five aggressor nets and a two-sink victim net with two supply wires as shields. As shown in the figure, with the insertion of shield s_1 and s_2, aggressor A_5 will not affect the victim net, and aggressor A_4 will also have less capacitive coupling with the victim net compared with the case when s_2 is absent.

In the global routing phase, the exact positions of signal nets are still undetermined, and hence neighborhood information is not fully available. At the same time, the switching activity of nearby signal nets is also unclear. In the absence of this detailed information, it is meaningful to determine a *worst-case* scenario based on the information that is available. We assume if one side of a signal net is not placed next to a shield, it will (pessimistically) be adjacent to an aggressor net that will induce coupling noise on the net. Thus a signal wire must have supply lines placed on both sides to be fully protected. However, due to limited routing resources, a net may not be fully protected. We refer to the number of protected sides of net k at a specific routing edge e_{ij} as $P_{ij,\,k}$, and it can take a value of 0, 1, or 2. If $P_{ij,\,k}$ is known for the signal wires across edge e_{ij}, the number of supply wires required for shielding can be found as follows. If there are S signal wires requiring protection on a single side, and D signal wires requiring protection on both sides, we must have p power supply as shields to achieve the protection, and the minimum amount of p can be found by the following equation:

$$p = \begin{cases} \frac{S}{2} + D & \text{if } S \text{ is even} \\ \frac{S+1}{2} + D & \text{if } S \text{ is odd} \end{cases} \tag{1}$$

Note that one power supply wire can shield two sides, corresponding to signal wires on both sides, which actually "share" the protection from this supply wire, and the above equation (1) has already taken this sharing effect into consideration. It is easy to prove that an arrangement of supply and signal wires with the above numbers exists, so that the desired protection is feasible. The specific positions of the signal wires and supply wires will be handled by detailed routing tools. Since supply wires and signal wires share the routing resources, the capacity constraint of edge e_{ij} requires that

$$C_e \geq w_s \cdot s_e + w_p \cdot p_e \tag{2}$$

where C_e is the boundary capacity; w_s and w_p are the width (including the metal width and the spacing) of a signal wire and a supply wire, respectively; s_e and p_e are the number of signal wires and supply wires passing the boundary e, respectively.

2) Noise Calculation with Devgan's Metric: We assume that a *noise margin*, NM_{spec}, is specified for each gate or buffer input in the circuit, and represents the largest noise voltage that will avoid a circuit malfunc-

tion[1]. An net is represented using a standard segmented coupled RC model so that its equivalent circuit contains a tree of resistors with capacitors to ground and to adjacent nets. The nodes in the tree are assigned parental relationships, with nodes closer to the source preceding those away from it. Our noise reduction algorithm propagates a set of candidate solutions in the search space, and to facilitate this, we recursively define the noise margin for an internal node i as:

$$NM(i) = \min_{all\ child\ nodes\ j}(NM(j) - V_n(i \leftrightarrow j)), \quad (3)$$

where $V_n(i \leftrightarrow j)$ is the noise voltage induced between i and j. A net is *noise free* if at any driving node (i.e., the net source or any buffer output), we have:

$$I_n \cdot R_d \leq NM \quad (4)$$

Here, R_d is the gate driver resistance, and I_n is the induced noise. The noise current can be calculated and propagated using Devgan's metric. This is illustrated by an example in Figure 3, which shows a signal wire segment extending from the center of grid cell i to the center of its neighbor cell j. Let the per unit length coupling capacitance and resistance be C_c and R_e, respectively, and the aggressor voltage slew rate be μ. Since the lower side of the wire segment is shielded, only the upper aggressor will induce noise on this net, implying the following result according to Devgan's metric:

$$I_n(j) = I_n(i) + L_e \cdot C_c \cdot \mu \quad (5)$$

$$NM(j) = NM(i) - R_e L_e \left(\frac{1}{2} C_c L_e \mu + I_n(i)\right) \quad (6)$$

If the noise at j satisfies the constraint (4), the buffer will block the propagation of noise, and the noise margin at j' will recover to NM_{spec}, with the noise current $I_n(j')$ reset to 0 as well.

D. Problem Formulation

The formal statement of the problem is as follows. Given a tiling of a chip and the corresponding routing graph $G = (V, E)$, nets $N = \{n_1,\ n_2,\ ..., n_m\}$, the edge capacity C_{ij} for every edge $e_{ij} \in E$, and the buffer capacity B_i for each routing grid cell $i \in V$, the problem is to find a routing solution that:

1) determines the routes for each net on the routing graph,
2) satisfies the power wire density constraints, i.e., the average and minimum density MAN_i and MN_i of each block must be met,
3) determines the grid cells in which a net is to be buffered, subject to the buffer capacity constraint,
4) finds $P_{ij,\ k}$ over each edge e_{ij} in the routing of net k, subject to the edge capacity constraint (2),
5) satisfies noise constraint (4) for all nets.
6) ensures that the total amount of interconnects that can be driven by a buffer (gate) is at most M grid units.

[1]As described later in Section III-B, this noise margin can be selected by inflating the actually desired noise margin, so that it accounts for the over-estimation in Devgan's metric.

III. Validity of Devgan's Metric

Devgan's metric is known to provide an upper bound for the crosstalk noise in an RC circuit, but this upper bound is also known to potentially result in large over-estimates [18]. We will now examine the utility of this metric, despite its known inaccuracies, in the specific context of our simultaneous shield and buffer insertion algorithm. In our approach, a set of candidate solutions is propagated, and the calculated noise is used to prune some of these solutions in one of the following two ways:

- Comparing the noise value V_{n,s_1} of solution s_1 and V_{n,s_2} of solution s_2 to identify the relatively larger one.
- Comparing the noise value $V_{n,s}$ of a solution s against a specified noise margin NM_{spec}, and pruning the solution if $V_{n,s} > NM_{spec}$.

We claim that for the above purposes, we can still safely use Devgan's metric despite its pessimism, and we will justify this by arguing for the fidelity of the metric, as well as the employment of a noise margin inflation technique, which can compensate for its pessimism.

A. Fidelity of Devgan's Metric

In the context of determining the relative order of protection solutions by their noise value, we argue that it is the *fidelity* of a noise metric that is important rather than the accuracy. Fidelity of a noise metric for protection solution refers to the degree to which an optimal or near-optimal solution according to this metric will also be near-optimal according to an accurate noise metric. During solution pruning, if the noise metric employed has good fidelity, the relative order determined by this metric will have a high probability of being the same as that determined by an accurate simulation method, such as SPICE. Thus we can claim that solutions pruned under the noise metric are truly inferior to others in terms of noise level.

We verify the *fidelity* of Devgan's metric with a set of experiments that are conducted under design scenarios that are *similar* to those encountered for our problem. We randomly generate the pin locations of a circuit with several multiple-sink nets in a 6×6 grid, and then route the nets using AHHK algorithm [3]. The coupling capacitances are then extracted using the technology parameters that will be detailed later in Section V. Next, we randomly pick one net as the victim net and consider the other nets as aggressors, and assume all aggressor nets adversely switch at the same time while the victim net remains at a stable value. This experimental setup is aligned with the assumption we made in the problem formulation, so that the verification results can be validly applied to our algorithm. Under this scenario, we compute the coupling noises at the sinks of the victim net

Aggressor rise time	# nets in circuit	Ranking: All victim sinks			Ranking: Worst noise sinks			average overlap (# of grid lengths)
		# victim sinks	Average rank diff.	% error	# worst noise sinks	Average rank diff.	% error	
200 ps	20	234	23	9.8%	100	6	6%	9.59
100 ps	20	264	32	12.1%	100	11	11%	10.49
200 ps	40	279	16	9.3%	100	4	4%	13.37
100 ps	40	271	33	11.1%	100	8	8%	13.61

TABLE I

VERIFICATION OF THE FIDELITY OF DEVGAN'S METRIC.

using Devgan's metric and using a SPICE simulation. The above experiment is repeated 100 times, and then all of the victim sinks in these experiments are ranked according to their noise under Devgan's metric and under SPICE, respectively. Since a net has an average of 2.5 sinks under our experiments, the noise is computed at a total of about 250 sinks in these 100 experiments. The rank difference of each such sink under the two metrics is then determined.

Our experiments correspond to the following cases:

- The number of nets in each experiment is either 20 or 40: under these assumptions, we find that the average number of crosstalk adjacencies per victim net is around 10 grids and 13 grids, respectively, which indicates that there is a significant amount of coupling. The two cases correspond to low and high net density in a routing region.
- The input waveform, modeled as a saturated ramp from 0 to 1.8V, has a transition time of either 100 ps or 200 ps, which simulate different aggressor switching time. Here we assume a uniform transition time for all aggressor nets, which is consistent with the noise model used in our shield and buffer insertion algorithm.

This results in four combinations of (number of nets = 20 or 40) and (transition time = 100 ps or 200 ps), and the corresponding results that show the rank difference of each sink are listed in Table I. For all 250 or so sinks in the experimental setup, the average errors that corresponds to the distance in ranking (between Devgan's metric and under SPICE) are around $9 \sim 12\%$, which suggests that Devgan's metric has good fidelity in comparing crosstalk noise. On average, while comparing two structures, if one of them has lower crosstalk noise than the other under Devgan's metric, it is very likely to also demonstrate lower noise under a SPICE simulation. The fidelity is even higher for the worst noise victim sink ranking for the 100 sinks with worst noise values in victim net, where the error is only around $4 \sim 11\%$.

These results lead to an important conclusion: *on average, Devgan's metric has good fidelity, and can be used as an effective metric to compare noise levels in our shield and buffer insertion algorithm.*

B. Noise Margin Inflation

The second pruning technique in our algorithm compares the noise value estimated from Devgan's metric against a specified noise margin NM_{spec}. In this comparison, due to the pessimistic nature of Devgan's metric, a noise value can be over-estimated, and thus the corresponding eligible solution can be falsely pruned, so that our buffer and shield insertion algorithm may over-optimize and use more than enough resources to accomplish full protection, or under stringent protection resources, may result in false-failures. To compensate for this pessimism of Devgan's metric, in practice, we can apply a noise margin inflation technique, i.e., we can heuristically inflate the specified noise margin NM_{spec} to be NM_{inf} ($NM_{inf} > NM_{spec}$). If chosen carefully, the inflated noise margin as input to our algorithm will compensate for the over-estimation of Devgan's metric.

Although there is no strict mathematical proof about the accuracy of employing this inflation technique, its validity can be verified and supported by the statistical model detailed below. The inflated noise margin NM_{inf} can be estimated from NM_{spec} using a curve-fitting technique. If we denote a noise voltage evaluated under Devgan's noise metric as V_d, and the value of the same noise evaluated under a SPICE simulation (which is considered as the accurate noise level) as V_s, then we use a fitting function $V_d \approx a_0 V_s + a_1$, where a_0 and a_1 are constants. Therefore, the inflated noise margin can be obtained from the relationship as $NM_{inf} = a_0 NM_{spec} + a_1$.

To generate the coefficients a_0 and a_1, we first randomly generate nets in a circuit with the same technical parameters as in Section III-A. The number of nets in each circuit is 30 to simulate the congested routing scenario of a complex design, and the aggressor transition time is 200 ps, which is consistent with our experimental setup in Section V. The experiments are performed 100 times, and the noise value at each sink is evaluated with both Devgan's metric (obtained as V_d) and SPICE simulation (obtained as V_s). We plot the distribution of V_d vs. V_s as in Figure 4, and each point in the figure has a coordinate of (V_s, V_d) which corresponds to noise value evaluated under different metrics. A least squares fit is then performed to find the values a_0 and a_1, and the result is shown by the dashed line in the figure. It is noted that we eliminate all the noise points with $V_d > V_{dd} = 1.8$V in performing the curve fitting for the following reasons: (a) It is not physically possible for a noise value to be larger than V_{dd} in a RC interconnect model, and Devgan's metric is known to be over-pessimistic here. (b) In practice, we observe that for noise with $V_d > V_{dd}$, the corresponding

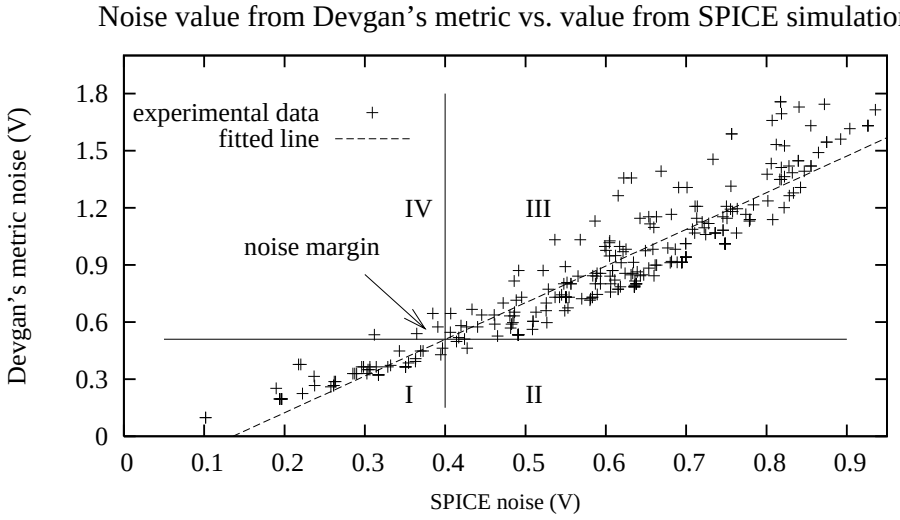

Fig. 4. Noise estimation under Devgan's metric and SPICE simulation. The least squares fit to a linear approximation between the two noise metrics is drawn as a dashed line. The plane is divided into four representative regions, I through IV, as shown above.

V_s is always much larger than a practically specified noise margin, and this eliminates the necessity to include them in the sample data. From the fitting results, we can obtain the values $a_0 = 1.93$ and $a_1 = -0.26$V under our experimental setup.

To illustrate the usefulness of the noise margin inflation technique, we draw a vertical line at a given noise margin point, and a horizontal line at the point where this vertical line crosses the fitting line. These horizontal and vertical lines divide the plane into four regions labeled as I, II, III and IV. We will use N_i, i = I, II, III, IV to indicate the number of points falling in each region i. Figure 4 shows an example of such crossing lines and the corresponding divided regions under a specified noise margin $NM_{spec} = 0.4$V. Points falling in region I [III] represent the cases in which a noise satisfies [violates] both the NM_{spce} under SPICE simulation and NM_{inf} under Devgan's metric. As shown in Figure 4, most of the points fall in regions I and III, which indicates that under our noise margin inflation technique, the qualitative noise value from Devgan's metric agrees well with the SPICE simulation result. For the noise margin inflation to be an effective technique, we desire that it should capture all the valid solutions, and thus a valid solution under SPICE simulation will not be falsely discarded under inflated noise margin. This effectiveness is reflected by the very small number of points in region IV, which are the points violating NM_{inf} under Devgan's metric but actually satisfying NM_{spec} under SPICE simulation. In the above example, only 1.7% (calculated as $N_{IV}/(N_{III} + N_{IV})$) of the total discarded noise values with our technique are actually good solutions but falsely pruned. Interpreted alternatively, our noise inflation technique can capture 96.4% (calculated as $N_I/(N_I+N_{IV})$) of all the eligible solutions in a statistical sense. At the same time, for noise inflation to be a safe technique, we also want to avoid false acceptance of a noise-failure solution, i.e., the protection solution obtained under the inflated noise margin NM_{inf} should also satisfy the original noise margin requirement NM_{spec}. This is validated observing that a very small number of points fall into region II in Figure 4. This safety requirement is crucial for the accuracy of our method, and we will further experimentally show in Section V that the noise margin inflation technique embedded in our simultaneous shield and buffer insertion algorithm does generate valid noise protection solutions, and the above proposed curve-fitting technique to estimate inflated noise margin is an accurate approach in practice.

IV. Routing and Crosstalk Reduction Algorithm

Our approach to the problem formulated in Section II-D is depicted in Figure 5. The algorithm is iterative and proceeds through three steps: congestion-driven global routing, a dynamic-programming-like simultaneous buffer and shield insertion procedure and rip-up-and-reroute refinements. Steps 2 and 3 iterate until all constraints are satisfied, or no further improvement is possible. In the above buffer and shield insertion and rip-up-and-reroute, we process one net at a time and maintain a fixed order of all nets. We have experimentally found under different randomly chosen net orderings, the results change very little, as long as we maintain the same fixed net order through all of the iterations. This is due to the fact that the early iterations are seen to create good estimates of resource utilization, and this reduces the order dependence.

A. Step 1: Congestion Driven Routing

The signal wire routing procedure consists of two phases, similar to that in [2]. The first phase of routing

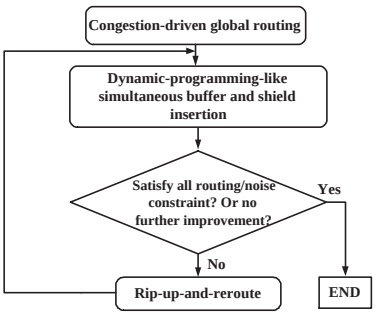

Fig. 5. Overall flow of global routing with simultaneous shield and buffer insertion for crosstalk noise reduction.

constructs Steiner trees using the AHHK algorithm [3], and works as a fast estimator of the congestion map. The second phase performs a congestion-driven rip-up-and-reroute based on this initial solution, with the objective of minimizing the congestion cost over routing grid edges [2]. If there is still an overflow violation after this phase, more rip-up-and-reroute steps will be performed (with the same net order), as in [21].

Besides punishing regular congestion overflow, the congestion cost function in the congestion-driven routing is designed to also incorporate the consideration of power supply requirements. Since all of the routing capacity is shared by signal and power routing, signal routing results will determine the power supply structure, and thus signal routing must leave enough capacity for power supply to satisfy the average and minimum power supply densities MAN_i and MN_i for a block i of the routing region. The cost function for a signal wire traversing an edge e in block i is composed of two terms:

$$\text{routing cost} = cost_e + cost_i \tag{7}$$

in which $cost_e$ is the cost of traversing edge e, and penalizes any violation of MN_i for edge e in block i; $cost_i$ is the cost of passing block i, and penalizes any violation of MAN_i of block i. Both terms take the following form:

$$cost = \begin{cases} \frac{U}{R} & \text{if } R > 0 \\ 10^{-R} & \text{if } R \leq 0 \end{cases} \tag{8}$$

where R is the residual signal routing capacity on the edge [block] for the first [second] term. This value is calculated by subtracting, from the total edge [block] capacity, the power supply requirements and the capacity already used for signal routing, U. Note for the first term, the power supply requirement is MN_i; for the second term, it is MAN_i multiplied by the number of edges in a block i, since MAN_i is defined *per edge*. The exponential form of the cost function after the capacity violation punishes the over-use of capacity from signal routing, so as the power grid requirements can be met. In addition, it can also effectively avoid the aggregation of signal wires.

The congestion driven routing performs a wave-form expansion starting from the source, and update tile cost with the lowest cost in each expansion like [3]. The routing of a net terminates upon all sinks have been reached. After every routing or rerouting, each of the grid cells in the final path is added to the tree as an *Internal Node (IN)*, and a net will then be a set $\{s \cup P \cup IN \cup E\}$, where *s* is the source node, *P* is the set of sink pins, *IN* is the set of internal nodes, and *E* is the set of edges; this data structure is used for the procedure in Step 2, which follows this.

B. Step 2: Buffer and Shield Insertion

With a routing solution from the above step, we simultaneously allocate shield and buffer resources to each net so that the solution can meet the noise requirement while using least protection resources. Each net is processed individually, and the procedure traverses the tree structure of the net in a bottom-up manner, starting from the sinks and moving towards the source, moving along one grid square at a time. Each tree is described in terms of nodes that correspond to the grid cells that it passes through. Assigning a direction to the tree from the source to the sinks, we refer to the grid cell that contains the immediate predecessor [successor] of a given node n in the tree as the parent [child] cell of the grid cell containing n. If a grid cell has more than two children, we can insert pseudo-nodes, so that the final tree is a binary tree for ease of later processing.

While traversing a net across a grid cell i, we have two methods for protecting it from crosstalk noise:

1) By deciding whether to insert a buffer or not at grid cell i: this corresponds to two possible buffer insertion configurations (0 or 1 buffer).
2) By protecting the net using supply shields on one side, on both sides, or choosing not to shield the net at all. If grid cell i is not the root of the tree, shield(s) may be inserted alongside the edge e_{ij} connecting current grid cell i and its parent grid cell j in the tree. This results in three possible configurations (0, 1, or 2 sides protected).

Therefore, in each bottom up step, we may have six possible configurations for the *protection structure*. However, we cannot locally determine at each grid point which scheme is globally optimal, and therefore an enumerative dynamic programming-like approach is adopted here in the same spirit as in Van Ginneken's algorithm [29].

1) Protection Cost and Solution Architecture: While traversing a net bottom-up, at grid cell i, we must find the protection cost corresponding to a protection structure, so as to measure the resource usage. For the protection cost related to buffer insertion, since the nets are processed one at a time, at any point in our insertion algorithm, the probability that an unprocessed net n_i crossing grid cell i will insert a buffer from i is $1/M$. Let p_i be the sum of these probabilities over all unprocessed nets crossing cell i, and the cost for insertion of $b_{req}(= 0$

or 1) buffer at a specific grid tile i is similar to that in [2]:

$$cost_{buf\ i}(b_{req}) = \begin{cases} 0, & \text{if } b_{req} = 0; \\ \frac{b_i + p_i + 1}{B_i - b_i}, & \text{if } b_{req} = 1, \\ & b_i + b_{req} \leq B_i; \\ \infty & \text{otherwise.} \end{cases} \quad (9)$$

This cost function will significantly increase the cost penalty as buffer resources become contentious. For shielding cost, we can calculate the number of power supply wires required based on the number of sides to be protected N_{sh} and equation (1). In the same spirit to punish contentious resource usage, the shielding cost $cost_{sh\ ij}$ for a signal wire can be obtained from a similar form of equation as (9) above, but the predicted shield usage takes a different approach: each unprocessed signal net will probabilistically have 1 side to be protected (assuming equal probability for $N_{sh} =$ 0, 1, or 2). Both the shielding cost and buffer cost are a measurement of the number of resources used, and they are approximately of the same order of magnitude. Therefore, we can combine them with a weighting factor λ (determined by resources availability) to develop a metric for resource usage, which we call the *protection cost* at grid cell i, denoted as PC_i:

$$\begin{aligned} &PC_i(b_{req}, N_{sh}) \\ &= \lambda cost_{buf\ i}(b_{req}) + cost_{sh\ ij}(N_{sh}) \end{aligned} \quad (10)$$

where j is the parent grid cell of grid cell i. This comprehensive cost function can be used as a metric to evaluate the noise protection resource usage at grid cell i. In our algorithm, the decision of buffer/shield insertion in grid cell i is not determined on a local choice, based on the value of PC_i; instead the most efficient noise protection structure is identified globally, i.e., the choice of buffer or shielding at each grid cell is selected so that a global sum of the cost function (10) is optimized. Our algorithm builds a noise protection solution for a net in a bottom-up traversal, and all feasible protection structures are maintained during this process. The protection cost PC of a protection structure is defined as the sum of all protection costs, PC_i, over the grid cells i that have been processed in the traversal. By the end of our procedure, a protection structure that has the minimum protection cost PC, while satisfying the noise requirement, will be selected. For each net, our algorithm selects the minimum resource usage solution for noise protection, and hence it can effectively resolve the contention for protection resources among nets.

At a cell i during the bottom-up traversal, the noise margin and noise current will vary according to the protection structure we choose, i.e., whether a buffer is inserted and the number of sides of the net that are shielded. If a buffer is inserted, the noise current will be "reset" to 0 and the noise margin set back to NM_{spec}. At each unbuffered location, depending on the number of sides of a signal wire that are shielded, we can have the following from equations (5) and (6):

$$\Delta I_n(N_{sh}) = (2 - N_{sh}) \cdot L_e \cdot C_c \cdot \mu \quad (11)$$

$$\Delta NM(N_{sh}) = R_e \cdot (\frac{1}{2} \cdot (2 - N_{sh}) L_e C_c \mu + I_n(i)) \quad (12)$$

where ΔI_n is the increase in the noise current due to the number of sides getting shielded $N_{sh} \in \{0, 1, 2\}$, and ΔNM is the noise margin decrease during the bottom-up step. To keep a record of the protection structure in the enumeration, we define a *protection solution* at routing cell i to be a 4-tuple $S = \{PC, NM, I_n, stru\}$, where NM is the noise margin at the end of the edge connecting grid cell i to its parent grid cell j, I_n is the noise current induced by the neighboring signal wire at the same point, and PC is the protection cost of the current solution. The last component, *stru*={*buffer,* N_{sh}}, represents the protection structure of the solution, where *buffer* is a binary number representing the number of buffers utilized at grid cell i, and N_{sh} is the number of sides (0, 1 or 2) on which the wire is protected.

To satisfy the constraint that the total amount of interconnect can be driven by a buffer (gate) is at most M grid cells, we maintain a *solution set array* (*SSA*) of length M at each grid cell. Each element in *SSA* is a set of solutions, and the array is indexed from 0 to $M - 1$. The solutions in $SSA[k], k \neq 0$ correspond to a total downstream interconnect of k grid cells to the nearest downstream buffer(s), and $SSA[0]$ stores the solution that correspond to the insertion of a buffer at the current grid cell.

2) Algorithm for Building Protection Solutions: A unitary step in the enumeration algorithm is to build the solution set array at the current grid cell based on the arrays at its child cell(s). Solutions in set $SSA[k],\ k \neq 0$ are propagated from solutions in the lower indexed sets of the children grid cells. During the process, we update the noise margin according to different shielding choices to form new solutions, and the least cost child solution will be used to build solution in $SSA[0]$ in which buffer is inserted right at current grid cell. The main procedure in the algorithm calls function `Find_sol_set_array` at the source cell, and returns the minimum cost solution that satisfies the noise constraint. The pseudo code for the algorithm is listed in Figure 6. Function `Find_sol_set_array` finds the solution set array for a grid cell. SSA is first initialized in line 1, and line 2 initializes the trivial case of a leaf cell , in which both noise current and protection cost are set to 0. Next, line 3 considers a single child cell and builds $SSA[i]$ from $SSA[i-1]$ of the child grid cell as the left part of Figure 7 shows, considering the three possibilities for the number of sides to be shielded: 0, 1 or 2. $SSA[M-1]$ of the child cell will be dropped to avoid violating the buffer load rule. In line 4, the protection solution with minimum protection cost is selected from the children solutions and a buffer is inserted. As a result, both the noise margin and noise current are recovered for this tuple. Line 5 deals with finding a solution set

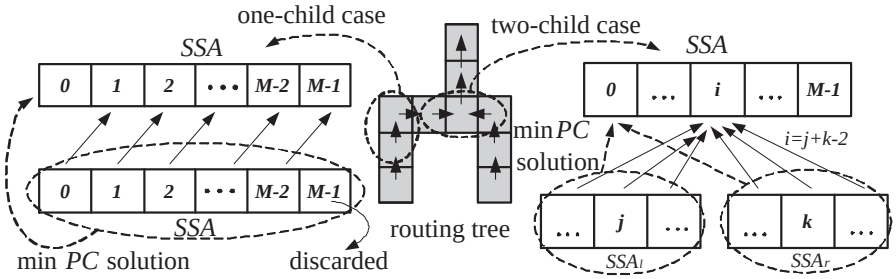

Fig. 7. Updating SSA for one-child and two-child grid cells.

Algorithm: `Shield_buffer_insertion_for_noise _reduction`

Input: Net $N = \{s \cup P \cup IN \cup E\}$
Output: A protection solution with least protection cost at source s

1) *SSA* = `Find_sol_set_array`(s)
2) return protection solution $S = \{PC, NM, I_n, stru\} \in SSA$, with PC is minimized.

Function: `Find_sol_set_array`

Input: t is the grid cell to be processed.
Output: The *SSA* of this grid cell.

1) $SSA[i] = \Phi$, $i = 0, 1, ...M-1$
2) **if** $t \in P$, is a leaf
   for i = 0 to $M-1$
   $SSA[i] = SSA[i] \cup \{0, NM_{spec}, 0, \Phi\}$;
3) **else if** t has one child l
   SSA_l=`Find_sol_set_array`(l);
   for i = 1 to $M-1$
   for each $S_l \in SSA_l[i-1]$
   $SSA[i] = SSA[i] \cup$ `Propagate`(S_l, t);
4) Take minimum PC solution S_m from SSA_l,
   $SSA[0] = SSA[0] \cup$ `Propagate`(`Insert_buf` $(S_m, t), t)$;
5) **else if** t has two children l and r
   SSA_l=`Find_sol_set_array`(l);
   SSA_r=`Find_sol_set_array`(r);
   for i = 2 to $M-1$
   for each $S_l \in SSA_l[j]$, $S_r \in SSA_r[k]$ and j+k=$i-2$
   $SSA[i] = SSA[i] \cup$`Propagate`(`Merge` $(S_l, S_r), t)$;
6) Take min $S_{ml}.PC + S_{mr}.PC$ solutions S_{ml}, S_{mr} from SSA_l, SSA_r;
   $SSA[0] = SSA[0] \cup$`Propagate`(`Insert_buf` (`Merge`(S_l, S_r), t), t);
7) Take minimum PC solution S_{ml} from SSA_l;
   for i = 1 to $M-1$
   for each $S_r \in SSA_r[i-1]$
   $SSA[i] = SSA[i] \cup$`Propagate`(`Merge`(`Insert_buf` (S_{ml}, t), S_r), t);
8) Take minimum PC solution S_{mr} from SSA_r
   for i = 1 to $M-1$
   for each $S_l \in SSA_l[i-1]$
   $SSA[i] = SSA[i] \cup$`Propagate`(`Merge`(S_l,`Insert_buf`(S_{mr}, t)), t);
9) Prune solution set array SSA;

Functions:
`Propagate`(S, t) /* Extend solution by one grid length upward*/
if t is source, **return** S;
else return $\{S.PC + PC_t(0, N_{sh}), S.NM - \Delta NM(N_{sh}), S.I_n + \Delta I_n(N_{sh}), S.stru \cup \{0, N_{sh}\}\}$, $N_{sh} = 0, 1, 2$;

`Insert_buf`(S, t) /* Insert a buffer to existing solution S */
return $\{S.PC + PC_t(1, 0),\ NM_{spec}, 0,\ S.stru\ \cup\ \{1, 0\}\}$;

`Merge`(S_l, S_r) /* Merge two solutions */
return $\{S_l.PC + S_r.PC, min(S_l.NM, S_r.NM), S_l.I_n + S_r.I_n, S_l.stru \cup S_r.stru\}$;

Fig. 6. Algorithm for building protection solution.

array for a grid cell with two children. The complexity involved here is that when we build the solution, all of the combinations must be considered, as the right part of Figure 7 suggests. Line 6 picks the least cost solution from both the left and right children, and inserts a buffer to drive both children, thereby recovering the noise margin and noise current. However, a buffer can also be inserted only to the left or right branch. Lines 7 and 8 build solution tuples that correspond to these two cases, where a buffer is inserted to only one child branch and combined with solutions from the other. In the above steps, functions `Propagate`, `Insert_buf` and `Merge` are called to build solutions. There can be a large number of solutions in the SSA for a grid cell after the `Propagate` and `Merge` steps; however, it is not necessary to keep all of them, and the solution pruning in the last steps will greatly reduce the solution set size. There are three types of pruning techniques employed here:

- We prune those solutions that violate the noise constraint (4).
- We prune the solutions that violate the buffer or wiring capacity.
- If a solution S_1 is provably *inferior* to another solution S_2 in the same SSA of a grid cell, i.e., if $PC_1 > PC_2$, $NM_1 < NM_2$ and $I_{n1} > I_{n2}$, then it is pruned from the set of solutions.

While there is no concrete way of proving that the size of SSA will be small, the pruning techniques work efficiently in practice; our experiments show that the number of solutions at each grid cell is limited between 3 and 60, and is less than 15 in most cases.

C. Step 3: Refinement

After the simultaneous shield and buffer insertion for noise reduction, refinement steps are applied if there are still some nets cannot be protected from noise constraint violation. The procedure is similar to that used in global routing phase. We rip-up and reroute all of the nets in the same fixed order as before. After one net is ripped up, it is rerouted immediately by the rerouting algorithm described in Section IV-A. However, the cost function in rerouting is now the combination of both the wiring congestion cost and the buffer congestion cost. This will drive the net to go through regions where wiring capacity and buffers are abundant. The dynamic programming-like algorithm for simultaneous shield and buffer insertion is then applied. After all of the nets

have been ripped up, rerouted, and then protected, the whole refinement step will be performed again if there is still some noise violation. The iteration stops until noise violations are fixed or there is no further improvement. In practice, the algorithm will not iterate many times, and our experimental results shown that there will not be much improvement after the third iteration. To provide additional protection from noise, in the last step, the unprotected nets will greedily take all of the unused wiring and buffer capacities along its path; however, this step is optional.

D. Algorithm Complexity

The algorithm complexity mainly consists of two parts: the complexity of the routing and rerouting algorithm, and the complexity of the simultaneous shield and buffer insertion algorithm. In practice, these two parts of the algorithm only iterate a limited number of times before stop, and therefore the number of iterations will only contribute a constant factor to the overall complexity. The routing and rerouting process essentially follows the maze routing algorithm, and thus has a complexity of $O(NGlogG)$, in which G is the total number of grid cells in the routing graph and N is the number of nets in the circuits. The computational complexity of our algorithm is dominated by the simultaneous shield and buffer insertion algorithm. For a single sink net, as function `Propagate` in Figure 6 suggests, the number of solutions will increase by 3 times for every bottom-up propagation. There are M solution sets, and all of the solutions in solution set $SSA[i]$ are actually derived from $SSA[i-1]$ of the child grid cell, with $SSA[M-1]$ of the child cell will be dropped. So the maximum number of solutions in a solution set can be 3^{M-1}, which is a constant. However, if there are $d > 1$ sinks in the tree, as line 5 in function `Find_sol_set_array` suggests, the maximum number of solutions can be as many as $O(C^d)$, where C is a constant. However, this is found to be unduly pessimistic in practice, since a large number of propagated solutions will be discarded as the pruning techniques works well. This also indicates why the approach is called "dynamic programming-like" instead of "dynamic programming": we have no provable optimal substructure property. However, our experiments show that the pruning method is efficient and the number of unpruned solutions at each node is small, as mentioned at the end of Section IV-B.2. Thus the running time for processing one net will not explode exponentially; instead, it will be asymptotically linear with the net length. Experimentally, we also observe that the asymptotic total runtime increases linearly with the number of nets in the benchmark circuits.

V. Experimental Results

A. Experimental comparisons

Our algorithm is implemented in C++ on a Linux PC with a 2.8GHz CPU and 1GB memory. Out of the 16 benchmarks, the first 14 benchmarks in Table II are 8 MCNC benchmarks and 6 GSRC benchmarks (*n30c, n50c, n100a, n200a, n200b, n300*). The largest benchmarks *syn1* and *syn2* with over 10,000 nets are randomly generated[2]. We superimpose a grid over the floorplan so that the geometry of each grid cell is almost a square. We assume that layer assignment has already been performed for the wires, and our router handles each layer separately. For simplicity, our results show one horizontal and one vertical layer, but the approach is easily extended to multiple layers. As there are no existing benchmarks that provide buffer information, we randomly generate the number of available buffers in each grid cell to capture the nature that such resource is reserved by block designers in the early design phase. The area of available buffers is set to be under three percent of total chip area. At the same time, we also control the buffer distribution to differentiate the performance of our algorithm from the other algorithms that we show comparisons with. We divide the design into several blocks, which correspond to different styles of circuits, such as control logic, data path, etc., and in our experiments, we use 7 blocks. The power supply requirements MAN_i and MN_i are randomly generated but in a balanced manner across the chip (in practice, these will be dictated by the functional blocks). We also assume that a power grid wire is twice the width of a signal wire. The routing edge capacities are assigned as shown in the table, in the units of signal wire width. We assume that the grid length $L_e = 600\mu$m, that for all gates, the noise margin specification $NM_{spec} = 0.4$V under a V_{dd} of 1.8V, and that the aggressor voltage change rate $\mu = 9 \times 10^9$V/s. The technology parameters used in the experiments are derived from [6] and [37] for the 0.18 μm technology: unit length coupling capacitance C_c=0.0583fF/μm, unit length resistance R_e=0.373Ω/μm, and buffer driver resistance R_d=180Ω.

We compare our results with those of two other methods. The first method is similar to the idea of [1], in which *only* buffers are inserted to reduce the noise, and the shielding effects are not considered. The buffers are also inserted by a dynamic programming-like algorithm, trying to achieve the noise constraint with the fewest number of buffers. The second method that we compared against is a two-step greedy approach, in which buffers and shields are assigned in separate stages. Buffer insertion is first performed in the same way as [2]. With the buffer positions known, we then attempt to greedily insert shield wires for each routing edge. For each net, the greedy shield insertion is composed of two steps:

1) We use a bottom-up approach, using every possible shield along the routing edges to meet the noise

[2]We did not use the ISPD98 placement benchmarks, because most of the nets in it are very short, which makes buffer insertion unnecessary, and cannot be used to illustrate the buffer contention problems that are projected for future technology nodes. This is consistent with the experience of the authors of [7].

Circuit	# of nets	Available buffers	EC	Grid	M	Average		# noise violation nets			Run time		
						MAN	MN	BS	G	B	BS	G	B
ami33	112	3011	9	30×33	6	3.2	1.5	0	31	46	5s	4s	8s
ami49	368	6889	16	30×33	7	4.2	1.5	0	66	103	12s	9s	20s
apte	77	1811	9	30×33	6	3.4	1.7	0	6	41	5s	3s	5s
hp	68	2386	9	30×33	5	3.5	2.2	0	14	21	3s	2s	4s
playout	1294	15884	56	30×33	7	18.0	5.5	0	165	568	51s	36s	69s
a9c3	1148	11847	44	30×33	6	13.0	5.6	0	106	481	37s	29s	50s
ac3	200	4034	14	30×33	6	4.6	2.4	0	20	85	9s	6s	12s
hc7	430	7938	26	30×33	7	7.8	4.0	0	59	122	13s	10s	22s
n30c	390	5513	15	30×33	7	4.3	1.5	0	58	149	11s	3s	13s
n50c	515	8404	17	30×33	6	4.2	1.8	0	55	140	10s	4s	15s
n100a	885	8931	32	30×33	6	7.2	4.3	0	68	348	25	7s	25s
n200a	1585	13803	64	33×33	7	20.1	10.0	0	86	742	57s	14s	58s
n200b	1714	16903	65	33×33	6	19.6	9.4	0	278	850	63s	29s	48s
n300	1893	23295	68	33×33	7	19.9	8.4	0	159	879	70s	33s	54s
syn1	10086	87773	334	40×40	6	96.5	60.7	0	1057	4836	508s	331s	563s
syn2	10486	90577	348	40×40	6	102.8	61.3	0	1345	4963	637s	398s	558s

TABLE II

COMPARISONS OF ROUTING AND NOISE PROTECTION RESULTS. EC IS THE EDGE CAPACITY; BS REPRESENTS THE SIMULTANEOUS BUFFER AND SHIELD INSERTION ALGORITHM; G REPRESENTS THE GREEDY ALGORITHM; B REPRESENTS THE BUFFER-ONLY ALGORITHM.

constraint.

2) If step 1 is successful, it may be the case that more than enough shields have been inserted. We then follow a top-down peel-off procedure to remove all of the unnecessary shields and buffers until the noise constraint or driving length constraint has been violated. The peel-off is greedy in the sense that a shield that is closer to the source of a tree will be removed greedily first. If there are multiple choices at any step in the top-down process, the branch with the higher noise margin will have its shield peeled off first.

Both of the above comparison methods employ the same routing and rerouting procedure as our algorithm.

The experimental results are listed in Table II. The first eight columns show some basic properties of the circuits. Next, the results of our method which introduces buffers and shields (BS), the first comparison method which introduces buffers only (B), and the greedy buffering and shielding method (G) are shown. Empirically, results show that the asymptotic runtime of our buffer and shield insertion algorithm is linear in the number of nets, and our algorithm scales easily to cases with over 10,000 nets. The B and G columns show that these methods can result in noise protection failures on as many as 53% (circuit $apte$) and 28% (circuit $ami33$) of the total number of nets. In comparison, our simultaneous shield and buffer insertion approach has achieved the protection goals successfully without much sacrifice in speed, and in all cases, all of the nets meet the noise constraints.

The buffer-only approach shows poor performance because of the restricted number of buffers that are available. In the competition for limited number of buffers, a large number of nets cannot obtain enough buffer resources. Without the help of intelligent shield insertion, they fail the noise protection. As interconnects continue scaling, next generation technology will see a more intensified contention for buffers, and this becomes more of an issue for future interconnect design, as projected by [25]. At the same time, the buffer-only algorithm also shows longer runtime than our BS algorithm for most of the testcases. This is due to that both our BS algorithm and the buffer-only algorithm follow the same rip-up-and-reroute procedure, which stops if a satisfactory solution is found or a specific number of tries have been reached. We find that our algorithm can reach a satisfactory solution before the prespecified number of tries, while for all the cases, the buffer-only approach must reach the maximum number of tries before it stops, but still fails. This explains why the buffer-only approach generally has a longer runtime. The greedy approach, on the other hand, performs buffer insertion and shield protection in separate steps, and no concerns of noise constraint are considered in buffer insertion, resulting in an inferior performance to our integrated buffer and shield insertion solution. However, its simple protection algorithm architecture makes it the most efficient algorithm in terms of runtime among the three.

Circuit	Overflow		Circuit	Overflow	
	BS	G		BS	G
ami33	0	416	*n30c*	0	271
ami49	0	583	*n50c*	0	398
apte	0	22	*n100a*	0	207
hp	0	168	*n200a*	0	168
playout	0	1774	*n200b*	0	4666
a9c3	0	981	*n300*	0	1604
ac3	0	228	*syn1*	0	30524
hc7	0	1054	*syn2*	0	36634

TABLE III

OVERFLOW OF ROUTING AND PROTECTION IF 100% PROTECTION IS ACHIEVED.

An additional advantage of our approach is the adaptive supply net architecture, which enables a flexibility between the requirements of signal and power routing, so that both routing and supply net requirements are simultaneously met. For all three algorithms in Table II, the routing overflow are almost 0 for all benchmarks, and is

hence not listed. However, in cases where more nets must be protected to resolve the remaining noise violations, extra routing resources must be employed, leading to overflows. Table III reports the overflow results for our algorithm and the greedy algorithm if 100% protection is desired (the buffer-only algorithm does not use shield resources, and is omitted from this comparison). The results show that much more routing resources have to be sacrificed to obtain a good protection for the greedy algorithm, while our algorithm can successfully achieve a good protection without extra routing overflow.

B. Verification of Noise Margin Inflation

As described in Section III-B, as an effective way to compensate for the pessimism of Devgan's metric, we can heuristically inflate the specified noise margin NM_{spec} to be $NM_{inf} \approx a_0 NM_{spec} + a_1$ and take this inflated value as the input to our algorithm. It is less pessimistic this way, and thus fewer protection resources are required to accomplish protection. Furthermore, we want to show experimentally that this inflated noise margin is at an appropriate level, so that it does not cause false positive errors during the pruning process in our algorithm, i.e., a protection solution satisfying the inflated noise margin NM_{inf} under Devgan's metric should also satisfy the originally specified noise margin NM_{spec} when measured with SPICE simulation.

The coefficients for noise margin inflation are experimentally determined in Section III-B with least square fitting technique and $a_0 = 1.93$ and $a_1 = -0.26$V. With $NM_{spec} = 0.4$V in our experiment, we can obtain $NM_{inf} = 0.51$V. Inputting NM_{inf} to our algorithm, the generated protection solutions are then simulated with SPICE to verify whether they still satisfy the originally specified noise margin $NM_{spec} = 0.4$V. Due to long runtime, we randomly selected up to 700 nets from each circuit for simulation; for smaller benchmarks such as *hp*, all nets were simulated. As a comparison, we further exaggerate the noise margin to $NM_{exag} = 0.6$V and input it to our algorithm. By performing the same SPICE verification process on the generated protection solutions, we compare the simulation results in Table IV.

In Table IV, P_{inf} is the percentage of nets that satisfy the inflated noise margin NM_{inf} under Devgan's metric, and also satisfies the specified noise margin NM_{spec}; P_{exag} is the percentage in the case for $NM_{exag} = 0.6$V. As can be seen, with NM_{spec} inflated to be $NM_{inf} = 0.51$V, almost 100% of the solutions can still satisfy the original $NM_{spec} = 0.4$V noise margin under accurate SPICE simulation; while this percentage drops to about 90% when NM_{spec} is further exaggerated to $NM_{exag} = 0.6$V. This result suggests that with too much inflation of noise margin, we can obtain false protection solution which actually fails the original noise requirement in reality; while our proposed methodology for noise margin inflation in Section III-B is safe and can acquire a good yet economic solution.

Circuit	P_{inf}	P_{exag}	Circuit	P_{inf}	P_{exag}
ami33	100%	93.3%	*n30c*	100%	88.9%
ami49	100%	92.4%	*n50c*	100%	86.5%
apte	100%	89.5%	*n100a*	100%	92.8%
hp	100%	94.3%	*n200a*	100%	90.7%
playout	100%	92.4%	*n200b*	100%	94.2%
a9c3	100%	95.3%	*n300*	100%	94.4%
ac3	99.3%	94.6%	*syn1*	100%	94.2%
hc7	99.1%	93.1%	*syn2*	100%	95.7%

TABLE IV

NET PROTECTION RATE WITH INFLATED NM_{spec}. P_{inf} AND P_{exag} ARE THE PERCENTAGE OF NETS GETTING FULLY PROTECTED UNDER SPICE SIMULATION WITH INFLATED NM_{spec} AT $NM_{inf} = 0.51$V AND $NM_{exag} = 0.6$V, RESPECTIVELY.

VI. CONCLUSION

We have shown in this paper a method for simultaneously inserting supply shields and buffers during global routing to reduce crosstalk noise under a novel power supply architecture. The method employs a length-constraint based buffer insertion model that also naturally takes into account delay considerations, and uses a dynamic programming-like bottom-up method to optimally assign shields and buffers with the least resource usage. We have also shown the fidelity of Devgan's metric, and the noise inflation technique to effectively compensate for its pessimism. Experimental results show that this method can route nets to meet both capacity and noise constraints. It is more effective than noise reduction using a buffer-only approach or a greedy approach.

REFERENCES

[1] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer Insertion for Noise and Delay Optimization," *IEEE Transactions on Computer-Aided Design*, 18(11), pages 1633-1645, November 1999.

[2] C. J. Alpert, J. Hu, S. S. Sapatnekar, and P. Villarrubia, "A Practical Methodology for Early Buffer and Wire Resource Allocation," *Proceedings of the ACM/IEEE Design Automation Conference*, pages 189-194, 2001.

[3] C. J. Alpert, T. C. Hu, J. H. Huang, A.B. Kahng, and D. Karger, "Prim-Dijkstra Tradeoffs for Improved Performance-Driven Routing Tree Design," *IEEE Transactions on Computer-Aided Design,* 14(7), pages 890-896, July 1995.

[4] M. R. Becer, D. Blaauw, I. Algor, R. Panda, C. Oh, V. Zolotov, and I. N. Hajjy, "PostRoute Gate Sizing for Crosstalk Noise Reduction," *Proceeding of the ACM/IEEE Design Automation Conference,* pages 954-957, 2003.

[5] Y. H. Cheng and Y. W. Chang, "Buffer Planning: Integrating Buffer Planning with Floorplanning for Simultaneous Multi-Objective Optimization," *Proceedings of the IEEE Asia and South Pacific Design Automation Conference,* pages 624-627, 2004.

[6] J. Cong, "Challenges and Opportunities for Design Innovations in Nanometer Technologies," *Proceedings of the International Symposium on Computing and Microelectronics Technologies,* 1998. (Available at: http://ballade.cs.ucla.edu/cong/papers/final1.pdf)

[7] J. Cong, T. Kong and, D. Z. Pan, "Buffer Block Planning for Interconnect-Driven Floorplanning," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design,* pages 358-363, 1999.

[8] J. Cong, D. Z. Pan, and P.V. Srinivas, "Improved Crosstalk Modeling for Noise Constrained Interconnect Optimization," *Proceedings of the IEEE Asia and South Pacific Design Automation Conference,* pages 373-378, 2001.

[9] A. Devgan, "Efficient Coupled Noise Estimation for On-chip Interconnect," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design,* pages 147-151, 1997.

[10] F. F. Dragan, A. B. Kahng, I. Mandoiu, and S. Muddu, "Provably Good Global Buffering Using an Available Buffer Block Plan," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design,* pages 358-363, 2000.

[11] W. C. Elmore, "The Transient Response of Damped Linear Network with Particular Regard to Wideband Amplifiers,"*Journal of Applied Physics,* 19(1), pages 55-63, January 1948.

[12] N. Hanchate and N. Ranganathan, "Post-Layout Gate Sizing for Interconnect Delay and Crosstalk Noise Optimization," *Proceedings of the International Symposium on Quality Electronic Design,* pages 92-97, 2006.

[13] N. Hanchate and N. Ranganathan, "A Linear Time Algorithm for Wire Sizing with Simultaneous Optimization of Interconnect Delay and Crosstalk Noise," *Proceedings of the International Conference on VLSI Design,* pages 283-290, 2006.

[14] P. Heydari and M. Pedram, "Capacitive Coupling Noise in High-Speed VLSI Circuits," *IEEE Transactions on Computer-Aided Design,* 24(3), pages 478-488, March 2005.

[15] T. Jing, L. Zhang, J. H. Liang, J. Y. Xu, X. L. Hong, J. J. Xiong, and L. He, "A Min-Area Solution to Performance and RLC Crosstalk Driven Global Routing Problem," *Proceedings of the IEEE Asia and South Pacific Design Automation Conference,* pages 115-120, 2005.

[16] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "An Exact Algorithm for Coupling-Free Routing," *Proceedings of ACM International Symposium on Physical Design,* pages 10-15, 2001.

[17] S. Khatri, A. Mehrotra, R. Brayton, and A. Sangiovanni-Vincentelli, "A Novel VLSI Layout Fabric for Deep Sub-Micron Applications," *Proceedings of the ACM/IEEE Design Automation Conference,* pages 491-496, 1999.

[18] M. Kuhlmann and S. S. Sapatnekar, "Exact and Efficient Crosstalk Estimation," *IEEE Transactions on Computer-Aided Design*, 20(7), pages 858-866, July 2001.

[19] P. B. Morton and W. Dai, "An Efficient Sequential Quadratic Programming Formulation of Optimal Wire Spacing for Crosstalk Noise Avoidance Routing," *Proceedings of the ACM International Symposium on Physical Design,* pages 22-28, 1999.

[20] P. B. Morton and W. Dai, "Crosstalk Noise Estimation for Noise Management," *Proceedings of the ACM/IEEE Design Automation Conference,* pages 659-664, 2002.

[21] R. Nair, "A Simple Yet Effective Technique for Global Wiring," *IEEE Transactions on Computer-Aided Design,* 6(2), pages 165-172, March 1987.

[22] L. W. Nagel, "SPICE2, a Computer Program to Simulate Semiconductor Circuits," University of California, Berkeley, CA, Technical Report ERL-M520, May 1975.

[23] L. T. Pillage and R. A. Rohrer, "Asymptotic Waveform Evaluation for Timing Analysis," *IEEE Transactions on Computer-Aided Design,* 9(4), pages 352-366, April 1990.

[24] P. Saxena and S. Gupta, "Shield Count Minimization in Congested Regions," *Proceedings of the ACM International Symposium on Physical Design*, pages 78-83, 2002.

[25] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick, "The Scaling Challenge: Can Correct-by-Construction Design Help," *Proceedings of the ACM International Symposium on Physical Design*, pages 51-58, 2003.

[26] J. Singh and S. S. Sapatnekar, "Topology Optimization of Structured Power/Ground Networks," *Proceedings of the ACM International Symposium on Physical Design,* pages 116-123, 2004.

[27] D. Sinha and H. Zhou,"Gate-Size Optimization Under Timing Constraints for Coupling-Noise Reduction," *IEEE Transactions on Computer-Aided Design,* 25(6), pages 1064-1074, June 2006.

[28] H. Su, J. Hu, S. S. Sapatnekar, and S. R. Nassif, "Congestion-driven Codesign of Power and Signal Networks," *Proceedings of the ACM/IEEE Design Automation Conference,* pages 64-69, 2002.

[29] L. P. P. P. Van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay," *Proceedings of the IEEE International Symposium on Circuits and Systems,* pages 865-868, 1990.

[30] A. Vittal and M. Marek-Sadowska, "Crosstalk Reduction for VLSI," *IEEE Transactions on Computer-Aided Design,* 16(3), pages 290-298, March 1997.

[31] A. Vittal, L. H. Chen, M. Marek-Sadowska, K. P. Wang, and S. Yang, "Crosstalk in VLSI Interconnections," *IEEE Transactions on Computer-Aided Design,* 18(12), pages 1817-1824, December 1999.

[32] J. Xiong and L. He. "Full-Chip Routing Optimization with RLC Crosstalk Budgeting," *IEEE Transactions on Computer-Aided Design,* 23(3) pages 366-377, March 2004.

[33] J. Xiong and L. He, "Extended Global Routing with RLC Crosstalk Constraints", *IEEE Transactions on Very Large Scale Integration Systems,* 13(3), pages 319-329, March 2005.

[34] T. Xue, E. S. Kuh, and D. Wang, "Post Global Routing Crosstalk Risk Estimation and Reduction," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 302-309, 1996.

[35] L. Zhang, T. Jing, X. L. Hong, J. Y. Xu, J. J. Xiong, and L. He, "Performance and RLC Crosstalk Driven Global Routing," *Proceedings of IEEE International Symposium on Circuits and Systems*, pages 65-68, 2004.

[36] H. Zhou and D. F. Wong, "Global Routing with Crosstalk Constraints," *Proceedings of the ACM/IEEE Design Automation Conference*, pages 374-377, 1998.

[37] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors,* 2003. (Available at: http://www.itrs.net/reports.html)