Enhanced Network Flow Algorithm for Yield Optimization

Cyrus Bamji and Enrico Malavasi
Cadence Design Systems, Inc. - San Jose, CA, 95134

Abstract

A novel constraint-graph algorithm for the optimization of yield is presented. This algorithm improves the yield of a layout by carefully spacing objects to reduce the probability of faults due to spot defects. White space between objects is removed and spacing in tightly packed areas of the layout is increased. The computationally expensive problem of optimizing yield is transformed into a network flow problem, which can be solved via known efficient algorithms. Yield can be improved either without changing the layout area, or if necessary by increasing the layout area to maximize the number of good chips per wafer. Our method can in theory provide the best possible yield achievable without modifying the layout topology. The method is able to handle a general class of convex objective functions, and can therefore optimize not only yield, but other circuit performance functions such as wire-length, cross-talk and power.

1 Introduction

One of the main sources of electrical failure in VLSI integrated circuits is the presence of *spot defects* [1], which can cause either extra material, or missing material at the place where the spot defect occurs. Modeling of the spot defect size distribution is used in the calculation of minimum design rules. The choice of the minimum design rules results from a trade off between area and yield considerations. Yield can be increased by using bigger feature sizes and wider spacings than the minimum required by the design rules. Therefore to improve yield it is beneficial to use non-minimum spacings whenever this does not introduce significant area penalty. Due to the very high costs associated with the manufacturing of sub-micron integrated circuits, even a modest yield improvement can be extremely significant. For instance in a modern deep-sub-micron foundry a 1% yield improvement can be worth over $10M per year [2].

At the layout design level, methods for improving yield due to spot defect failures fall into two broad complementary categories. In the first category the layout topology is changed to improve yield. This generally involves changing the routing of the components in the layout. In [3], [4] and [5], channel routing is modified to minimize the critical areas between wire segments, so that bridging defects are minimized. In [6], several types of faults due to spot defects are considered. Their distributions are used to build a cost function for a weight-driven area routing approach. Although yield optimization during routing has proved effective, routing is a constructive procedure, whose results are sensitive to net schedule.

In the second category of methods for yield improvement the layout topology is fixed. Components and routing are spaced to minimize the probability of faults due to spot defects. Such a spacing-based approach is described in [7], where a heuristic algorithm increases the spacing of layout objects through a series of spacing iterations. By changing the positions of only objects off the critical path, layout area is maintained at its minimum size. This heuristic however does not guarantee optimum yield. Also many industrial cases require simultaneous optimization of multiple objectives such as yield, wire-length and crosstalk. It is difficult to extend this approach to handle such simultaneous objectives, and to allow the chip area to increase if necessary to maximize the number of good chips per wafer.

In this work a new algorithm (*Enhanced Network Flow Algorithm*) for the optimization of yield via spacing is presented. The network flow algorithm is extended to allow fast optimization of a large class of convex functions such as yield, wire-length minimization and crosstalk. Because of the additive properties of convex functions, multiple objectives can be handled simultaneously. This paper focuses primarily on applying our algorithm to the optimization of yield via spacing. It can be used to optimize yield while maintaining fixed (i.e. minimal) area, or if necessary increasing chip area to maximize the number of good chips per wafer.

Our algorithm leverages off the data structures and methods commonly used in layout compaction. The problem of yield optimization via spacing is transformed into a network flow problem which is then solved using a fast wire-length minimization algorithm [8].

The fault probability between two adjacent objects in the layout is expressed as a convex decreasing function of their distance. The problem of yield optimization is then to minimize the sum of the fault probabilities for all pairs of adjacent objects in the layout. A constraint graph for the layout is maintained, where each object in the layout corresponds to a vertex, and an edge links the vertices of adjacent layout objects. The cost of each graph edge is the fault probability due to a spot defect between the two corresponding objects and can be expressed as a function of edge length. Such a function can be effectively approximated by a convex piece-wise linear cost function. The graph is first modified so that the network flow algorithm can be applied to it. The graph is modified by replacing each edge by a small *equivalent* sub-graph with as many vertices as there are segments in the piece-wise linear approximation. The cost on the edges in each sub-graph are linear functions of their length and consequently all edges in the modified graph have this property. Vertex positions of the modified graph which minimize the sum of the new edge costs can therefore be found using the network flow algorithm. In this work it is proved that after this operation, the position of the original vertices in the modified graph also minimizes the cost of the original graph. Hence a solution for the yield optimization problem is obtained.

The paper is organized as follows. First a rigorous explanation of the *Enhanced Network Flow Algorithm* is presented in Section 2 without any reference to its application to yield optimization. In Section 3 the yield model is introduced. In Section 3 the algorithm developed in Section 2 is applied to the model described in Section 3 to optimize yield for fixed and variable area. Experimental results are reported in Section 4.

2 The Enhanced Network Flow Algorithm

In this section, a method called *Enhanced Network Flow Algorithm* is presented. It extends the well known network flow algorithm [9, Ch. 27] to handle a more general class of problems. Throughout this paper a 1-D constraint graph is considered. Without loss of generality it is assumed that spacing is performed in the horizontal dimension.

Let $\mathcal{G}(E, V)$ be the constraint graph, where E is the edge set, and V is the vertex set. A directed edge from vertex a to vertex b in $\mathcal{G}$ is denoted $(a \rightarrow b)$. Its length is the difference between the location of vertex b and that of vertex a and is denoted $l(a \rightarrow b)$. Each edge $(a \rightarrow b)$ in the graph has a minimum length denoted by $l_{min}(a \rightarrow b)$. The cost of edge $(a \rightarrow b)$, denoted $c(a \rightarrow b)$, is a function which depends on the edge length. The cost of the graph is the sum of all the edge costs. If the edge cost $c(a \rightarrow b)$ is proportional to the edge length, the slope of the cost function is a constant called *weight* of edge $(a \rightarrow b)$, denoted by $w(a \rightarrow b) = c(a \rightarrow b)/l(a \rightarrow b)$. The location of vertex v is denoted $l(v)$. The constraint graph is assumed to have a single source v_L and a single sink v_R.

The size of the layout in the horizontal dimension (the dimension of spacing) is denoted by L_c, and its size in the opposite direction (its height) is denoted by H_c. The area of the layout is $A_c = (L_c \cdot H_c)$. The notation w, l_{min} near an edge as shown in Figure 1, indicates that the weight of the edge is w, and its minimum length (constraint) is l_{min}.

In [8], an efficient network flow heuristic algorithm is presented to solve the wire-length minimization (WLM) problem. This algorithm finds the minimum cost configuration of the graph provided all edge costs are proportional to the edge lengths. In this paper the network flow algorithm is extended to a more general case, where the edge cost is a piece-wise linear (PWL) convex function of the edge length.

Piece-wise linear convex cost function

For each edge $(a \rightarrow b)$, let $\lambda = l(a \rightarrow b)$ be its length. On a finite set of N_s coordinates $x_1, x_2, \ldots, x_{N_s}$, the values of a PWL cost function of edge $(a \rightarrow b)$ are known, and indicated as $F_1, F_2, \ldots, F_{N_s}$ respectively. The slope of function F between x_i and x_{i+1} is constant and finite, and is denoted by S_i. Function F is continuous and convex. Because of convexity its slope is non-decreasing:

$$S_1 < S_2 < \ldots < S_{N_s} \tag{1}$$

Function $F(\lambda)$ is not defined for $(\lambda < x_1)$, while for $(\lambda > x_{N_s})$ it has a finite constant slope equal to S_{N_s}. Function F is shown in Figure 2 for $N_s = 4$.

Sub-graph substitution

In order to apply the network flow algorithm, the original graph must be transformed into a graph where the cost of each edge is proportional to edge length and hence a constant weight can be defined for each edge. This is achieved by substituting each edge which has a PWL cost function with a sub-graph. All edges in this sub-graph have cost proportional to length, (i.e. they have a well defined weight).

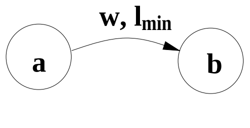

Figure 1: *The notation w, l_{min} indicates that the weight of the edge is w, and its constraint is l_{min}.*

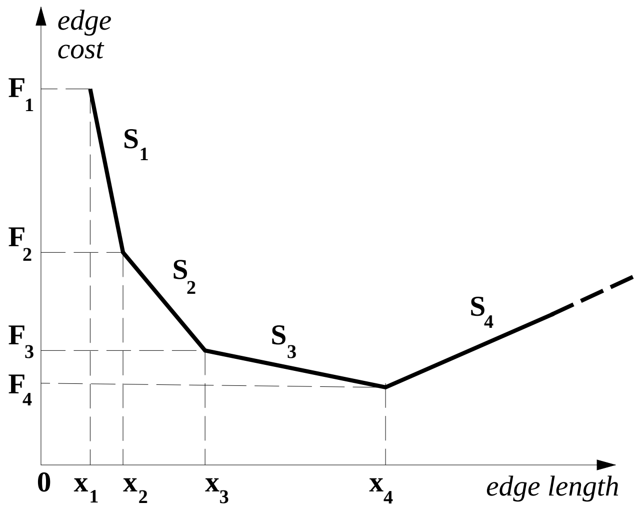

Figure 2: *Piece-wise linear cost function for edge $(a \rightarrow b)$, with $N_s = 4$.*

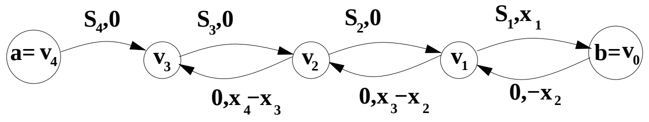

Figure 3: *Equivalent sub-graph for edge $(a \rightarrow b)$, with $N_s = 4$.*

Every edge $(a \rightarrow b)$ whose cost function is an N_s-segment PWL function is substituted by a sub-graph $G(a \rightarrow b)$ made of $(N_s - 1)$ vertices $v_1, \ldots, v_{N_s-1}$, and $2N_s - 1$ edges as shown in Figure 3 for the PWL cost function of Figure 2. The fanout and fanin vertices a and b of edge $(a \rightarrow b)$ are renamed respectively v_{N_s}, and v_0. Vertices $v_1, \ldots, v_{N_s-1}$ are called *implicit vertices* of sub-graph $G(a \rightarrow b)$. Sub-graph $G(a \rightarrow b)$ has the following properties:

- for each $i = 1, \ldots, N_s$, there is a directed edge $(v_i \rightarrow v_{i-1})$ (called *forward edge*) with weight S_i, and constraint:

$$l_{min}(v_i \rightarrow v_{i-1}) = \begin{cases} 0, & \text{if } i > 1 \\ x_1 & \text{if } i = 1 \end{cases}$$

- for each $i = 1, \ldots, N_s - 1$, there is a directed edge $(v_{i-1} \rightarrow v_i)$, (called *backward edge*), with weight 0, and constraint:

$$l_{min}(v_{i-1} \rightarrow v_i) = \begin{cases} -x_{i+1} - x_i, & \text{if } i > 1 \\ -x2, & \text{if } i = 1 \end{cases}$$

Sub-graph configuration

The length of edge $(v_i \rightarrow v_{i-1})$ in the sub-graph is denoted by $d_i = l(v_i \rightarrow v_{i-1})$. The array of all forward edge lengths $\mathbf{d} = [d_1\ d_2\ \ldots\ d_{N_s}]^T$ is called *configuration* of the sub-graph. Note that by construction, the substitution of sub-graph $G(a \rightarrow b)$ for edge $(a \rightarrow b)$ introduces no positive cycles in the constraint graph. Also, the minimum distance between vertices a and b imposed by sub-graph $G(a \rightarrow b)$ remains the same as that imposed by the original edge $(a \rightarrow b)$. The cost of the sub-graph is denoted by F_G and can be expressed in terms of its configuration $\mathbf{d}$:

$$F_G(\mathbf{d}) = \sum_{i=1}^{N_s} S_i \cdot d_i$$

If the locations of vertices a and b are not changed, the sum of the forward edge lengths must be equal to the length of the original edge:

$$\sum_{i=1}^{N_s} d_i = l(a \rightarrow b). \tag{2}$$

For every forward edge $(v_i \rightarrow v_{i-1})$, its length d_i has one of the following three possible states:

min extension: if d_i is equal to the edge constraint:

$$d_i = l_{min}(v_i \rightarrow v_{i-1})$$

max extension: if d_i is equal to the module of the constraint on the backward edge between the same vertices:

$$d_i = |l_{min}(v_{i-1} \rightarrow v_i)|$$

active state: if d_i is between the two constraints:

$$l_{min}(v_i \rightarrow v_{i-1}) < d_i < |l_{min}(v_{i-1} \rightarrow v_i)|.$$

Notice that edge $(v_{N_s} \rightarrow v_{N_s-1})$ can never reach maximum extension because there is no backward edge between vertices v_{N_s-1} and v_{N_s}. Therefore it can be stretched indefinitely.

Definition 1 *Given an edge $(a \rightarrow b)$, its sub-graph $G(a \rightarrow b)$ has* **minimum configuration**, *denoted* $\mathbf{d}^{(\mathbf{0})} = [d_1 \ldots d_{N_s}]^T$, *if* $\forall i \in [1, \ldots, N_s - 1]$:

- **If** d_i *does not have maximum extension* **then** d_{i+1} *has minimum extension.*
- **If** d_{i+1} *does not have minimum extension* **then** d_i *has maximum extension.*

For example, for the sub-graph $G(a \rightarrow b)$ of Figure 3 if $\lambda = l(a \rightarrow b)$ and $x_2 < \lambda \leq x_3$ then $\mathbf{d}^{(\mathbf{0})} = [x_2 \;\; (\lambda - x_2) \;\; 0 \;\; 0]^T$. In a minimum configuration there is always an integer k such that for all $i < k$ $(i \geq 1)$, d_i has maximum extension and for all $j > k$ $(j \leq N_s)$, d_j has minimum extension.

Figures 4.a and 4.c show the graph $G(a \rightarrow b)$ of Figure 3 in its minimum configuration for two different values of $\lambda = l(a \rightarrow b)$. The cost function of the original edge $(a \rightarrow b)$ is shown in Figure 4.b. As the original edge $(a \rightarrow b)$ is compressed the forward edges of $G(a \rightarrow b)$ starting from the left hand side of the graph are compressed and reach minimum extension. Only after an edge $(v_{i+1} \rightarrow v_i)$ reaches minimum extension does the next edge $(v_i \rightarrow v_{i-1})$ (which necessarily has lesser weight) begin to get compressed from its maximum extension. It can be easily shown that for any value of $\lambda = l(a \rightarrow b)$ the minimum configuration of $G(a \rightarrow b)$ is the configuration which has the lowest cost.

In Figure 4.a the leftmost segment has reached minimum extension and the second leftmost segment (with weight S_3) is in its active state. All other segments have maximal extension. The original edge length $\lambda = l(a \rightarrow b)$ is necessarily between x_3 and x_4. Further compression while maintaining $G(a \rightarrow b)$ in its minimum configuration will reduce the length of the second segment. Hence the change in the cost of the graph is $\triangle C_{G(a \rightarrow b)} = S_3 \triangle \lambda$. This is equal to the change of the cost function of Figure 4.b corresponding to a length variation $\triangle \lambda$ between x_3 and x_4 (in that region the slope of the cost function is S_3). Similarly in Figure 4.c the two leftmost segments have reached minimum extension and the third leftmost segment (with weight S_2) is in its active state. The change in cost for a length variation $\triangle \lambda$ is $\triangle C_{G(a \rightarrow b)} = S_2 \triangle \lambda$ which tracks the slope of the cost function of Figure 4.b between x_2 and x_3.

It can be shown that for any length $\lambda = l(a \rightarrow b)$ the cost of graph $G(a \rightarrow b)$ in its minimum configuration and the original edge cost $c(a \rightarrow b)$ differ by the constant $K = F_1 - S_1 x_1$. The proof involves some algebra and is omitted here for readability and lack of space.

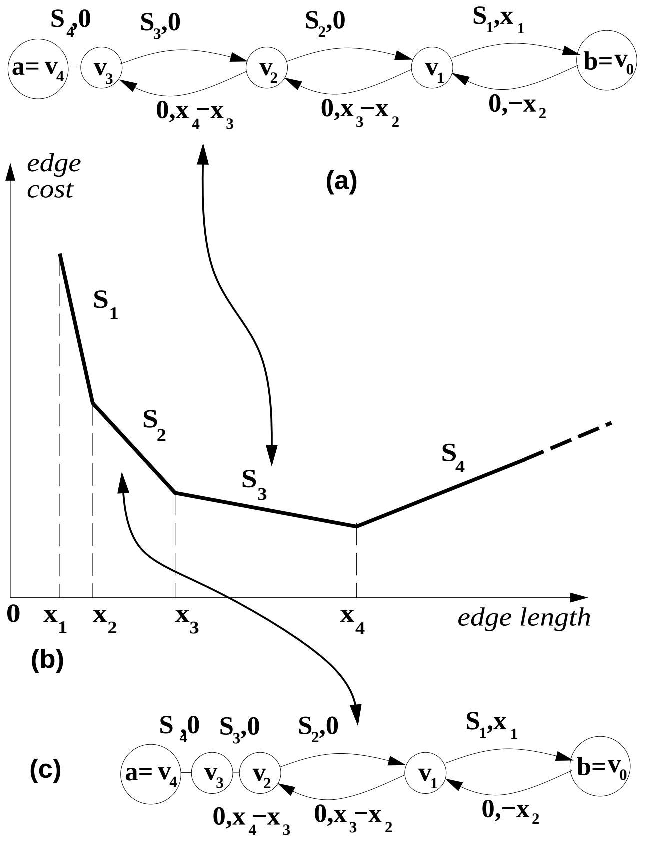

Figure 4: *Cost function $c(a \rightarrow b)$ and corresponding minimum graph $G(a \rightarrow b)$ configurations.*

Claim 1 *The set of locations of non-implicit vertices which minimize the cost of the modified graph $\mathcal{G}'$, also minimizes the cost of $\mathcal{G}$.*

Proof 1 *If $\mathcal{G}'$ has minimum cost all the substitution sub-graphs of $\mathcal{G}'$ are in their minimum configuration because the lowest cost for each substitution sub-graph occurs in the minimum configuration. Also the cost of $\mathcal{G}'$ and $\mathcal{G}$ (with the same vertex locations as the non implicit vertices of $\mathcal{G}'$) differ by a constant. Therefore the minimum for $\mathcal{G}$ and $\mathcal{G}'$ occur for the same locations of the non implicit vertices and hence $\mathcal{G}$ has minimum cost.*

Enhanced network flow algorithm

The enhanced network flow algorithm is based on the result of Claim 1 and involves the following steps.

1. In the constraint graph $\mathcal{G}$, every edge is substituted by the corresponding sub-graph structure shown in Figure 3 yielding a new graph $\mathcal{G}'$.
2. The network flow algorithm (in our implementation the WLM algorithm of [8] is used) is applied to $\mathcal{G}'$.
3. The locations of the non-implicit vertices in $\mathcal{G}'$ are used as the solution to the original problem.

Example: WLM with Jog Swapping

This section illustrates the use of the ENFA algorithm on an interesting special case. Consider the wire shown in Figure 5.a. The two vertical wire segments A and B are associated with two vertices in the constraint graph, linked

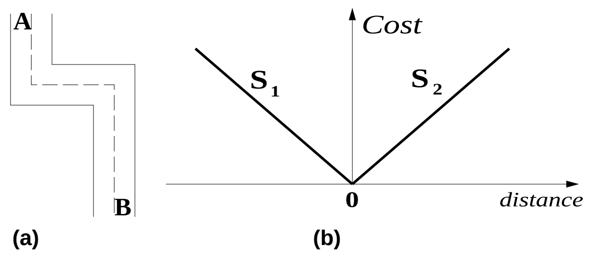

Figure 5: *(a) Wire jog and (b) its cost function.*

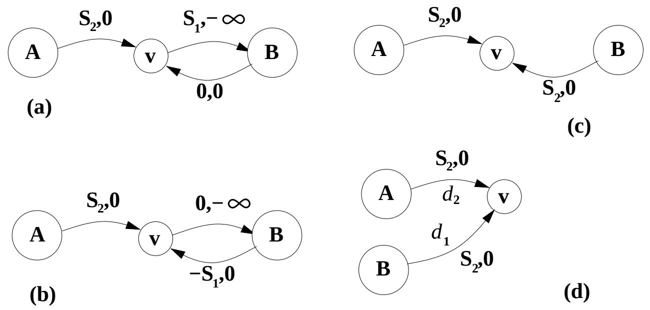

Figure 6: *(a) The equivalent sub-graph; (b) Weight S_1 is moved to another edge; (c) Useless edge is removed and $S_1 = -S_2$; (d) The final sub-graph.*

by an edge with constraint $-\infty$ (no constraint). The cost of the edge is proportional to the length of the horizontal wire segment connecting A and B, and is a function of the relative position of B with respect to A shown in Figure 5.b. The ordinary network flow algorithm is not able to handle this case and most wire length minimization algorithms treat this as a special condition. However the cost function of Figure 5.b is piece-wise linear, convex, with $N_s = 2, x_1 = -\infty, x_2 = 0, S_1 = -S_2$, and hence can be dealt with by the ENFA algorithm.

The edge linking A and B is substituted by the sub-graph shown in Figure 6.a. This sub-graph is equivalent to the one in Figure 6.b where weight S_1 on edge $(v \rightarrow B)$ is transformed into a weight $-S_1 = S_2$ on edge $(B \rightarrow v)$. Edge $(v \rightarrow B)$ then has zero weight and no constraints. Therefore it can be removed from the sub-graph as shown in Figure 6.c. The final equivalent sub-graph is as shown in Figure 6.d. Its cost is:

$$F_G = S_2 \cdot (d_1 + d_2)$$

Because $S_2 > 0$ from Figure 6.d it is clear that the cost of the sub-graph is minimized by minimizing the location $l(v)$ of v. This corresponds to the minimum configuration of the sub-graph. The minimum location $l_{min}(v)$ for v is $l_{min}(v) = \max(l(A), l(B))$ where $l(A)$ and $l(B)$ are the locations of vertices A and B. Therefore if $l(A) \geq l(B)$, in the minimum configuration $l(v) = l(A)$. Hence the cost of the sub-graph is $S_2 \cdot d_1$, where $d_1 = l(A) - l(B)$ is the length of the horizontal wire segment. Similarly if $l(A) \leq l(B)$ the cost of the sub-graph is $S_2 \cdot d_2$, where $d_2 = l(B) - l(A)$ is again the length of the horizontal wire segment. The cost of the minimum sub-graph configuration is thus the cost function of Figure 5.b. Finding values for $l(A), l(B)$ and $l(v)$ that minimize the cost of the sub-graph does indeed minimize the length of the horizontal wire segment.

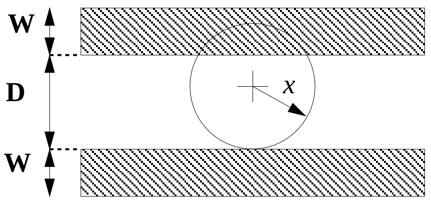

Figure 7: *Bridging fault of type OE between parallel wires spaced by D.*

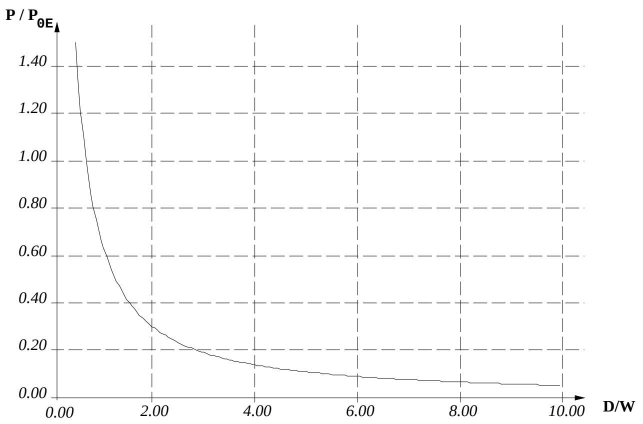

Figure 8: *Probability of a fault of type OE between wire segments spaced by D.*

3 Yield optimization

Yield model

The distribution of spot defects that occur in a manufacturing process is a function of the defect size x and is often modeled using the following expression [10]:

$$D(x) = \frac{X_0}{x^3}$$

where X_0 is a process-dependent constant.

Using the notation introduced in [6], a fault due to a spot defect bridging two parallel wire segments on the same layer (see Figure 7) is called fault of type *OE* (One-layer Extra-material defect). The probability of a fault of type OE between two parallel wire segments at a distance D from each other is [6]:

$$P = P_{OE} X_0 \cdot \left(\frac{1}{D} - \frac{1}{2D + W} \right) \qquad (3)$$

where W is the average width of the two segments. The constant P_{OE} is proportional to the length L of the parallel portion of the two segments [11]. A plot of the fault probability function P of equation (3) is shown in Figure 8.

From expression (3), it is clear that yield benefits from increasing the distance between any pair of parallel wire segments in the circuit. Every edge $(a \rightarrow b)$, between nodes representing two parallel wire segments on the same layer at distance D, has a cost equal to the probability of a fault of type OE. The cost function is:

$$C(a \rightarrow b) = P_{OE} X_0 \cdot \left(\frac{1}{D} - \frac{1}{2D + W} \right) \qquad (4)$$

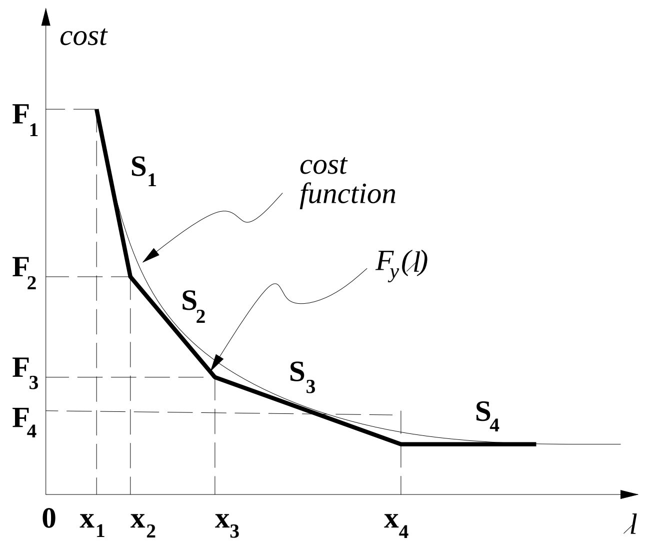

Figure 9: *Piece-wise linear function approximating the cost function accounting for the probability of faults due to spot defects, with $N_s = 4$.*

Cost function (4) is expressed in terms of the length λ of edge $(a \to b)$, which is equal to D plus some constant offset. Function (4) is a convex function of λ, therefore a reasonable approximation satisfying property (1) can be found. The edge cost function is approximated by a piece-wise linear (PWL) function $F_y(\lambda)$ with N_s points $x_1, \ldots, x_{N_s}$. The number N_s of points, their coordinates and the values of $F_y(\lambda)$ at those points are chosen to maintain the difference between the slopes of the fault probability function (4) and the yield cost function $F_y(\lambda)$ within an arbitrarily small predetermined value. For most practical cases good approximations are obtained with three to four segments in the approximation. The PWL approximation $F_y(\lambda)$ of the original cost function is shown in Figure 9 with $N_s = 4$. The minimization of the PWL cost function is equivalent to optimizing the contribution to yield due to spot defect faults of type OE.

Yield optimization with minimum area

Yield can be optimized while maintaining minimum area. Let $\overline{L}_c$ be the minimum layout size (in the spacing direction) achieved with one-dimensional compaction on the original constraint graph. $\overline{L}_c$ is the longest path from v_L to v_R and can be computed using the Bellman Ford Algorithm [9, Ch.25.3]. In the constraint graph, a directed edge is inserted from v_R to v_L with weight 0 and constraint $-\overline{L}_c$, as shown in Figure 10 (edge 1 is present, but edge 2 is not). The enhanced network flow algorithm is applied to the new graph thus obtained. Since the new edge keeps source and sink at their minimum distance, yield is optimized at no area cost by correctly spacing only objects off the critical path. The underlying network flow problem can be solved using the fast WLM algorithm in [8].

Yield optimization with variable area

If area is allowed to increase, the constraint on the maximum distance between graph source and sink is removed. Instead of optimizing yield with minimum area now the optimization objective is to maximize the number of good chips per wafer, which is approximately proportional to Y/A_c, where Y is the yield and A_c is the chip area. This problem is more

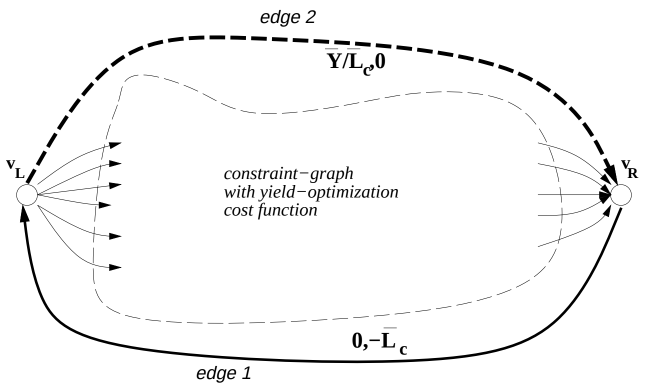

Figure 10: *Constraint graph for maximum yield. For minimum area source and sink are constrained to lay at their minimum distance $\overline{L}_c$ determined by the compactor (edge 1). Otherwise a directed edge is added between source and sink, with weight $\overline{Y}/\overline{L}_c$ (edge 2).*

complicated than in the minimum area case because now the overall cost of increasing an edge length $l(a \to b)$ depends not only on the edge costs but also whether the edge is on the critical path or not. Increasing the length of edges not on the critical path affects only Y. Increasing the length of edges on the critical path affects not only Y but also A_c because the area of the layout increases.

Let $\overline{Y}$ and $\overline{A}_c$ be respectively the nominal values of yield and the chip area. These values can be for example the minimum area and the best yield for the minimum area case. During optimization both area and yield change with respect to their nominal values:

$$Y = \overline{Y} + \Delta Y\ , \qquad A_c = \overline{A}_c + \Delta A_c.$$

The cost of the chip can be expressed as

$$\text{Cost} = \frac{A_c}{Y} = \frac{\overline{A}_c + \Delta A_c}{\overline{Y} + \Delta Y}$$

since it is assumed that $\Delta Y \ll \overline{Y}$ and $\Delta A_c \ll \overline{A}_c$. And since $\Delta Y = -\Delta F_y$ (where F_y is the cost of the original graph),

$$\frac{A_c}{Y} \approx \frac{\overline{A}_c}{\overline{Y}} \left[1 + \frac{\Delta A_c}{\overline{A}_c} + \frac{\Delta F_y}{\overline{Y}} \right] \tag{5}$$

Hence since $A_c = L_c \cdot H_c$, the cost function (5) changes by

$$\Delta \frac{A_c}{Y} = \frac{\overline{A}_c}{\overline{Y}^2} \left[\frac{\overline{Y}}{\overline{L}_c} \Delta x + \Delta F_y \right]$$

Therefore function (5) can be minimized by solving a modified graph, obtained from the original constraint graph by adding an edge between the source v_L and the sink v_R, with weight $w(v_L \to v_R) = \overline{Y}/\overline{L}_c$, as shown in Figure 10 (edge 2 is present, but edge 1 is not). The cost of this new graph is $F_y + (\overline{Y}/\overline{L}_c) \cdot x$, where x is the length of the additional edge.

Algorithm for yield/area optimization

The yield optimization algorithm is as follows:

1. Add a directed edge to the graph, between source and sink, with weight $\overline{Y}/\overline{L}_c$ and $l_{min} = 0$.
2. Solve this modified graph with the enhanced network flow algorithm described in Section 2.

The computational cost of this algorithm is practically the same as running the algorithm for the fixed area case, because the number of edges is the same, and only one edge differs in terms of its direction, weight and constraint.

			Fault	CPU[†] (sec.)	
name	des. style	size	reduct.	yld opt.	total
lev4	custom	10K	12.4%	4	11
bpr10	macro-cell	20K	12.7%	13	30
bpr12	macro-cell	6K	8.2%	1	3
bpr14	std. cell	1K	18.2%	< 1	< 1
bpr21	macro-cell	6.5K	17.9%	1	7
bpr22	macro-cell	6K	8.6%	1	6
bpr23	macro-cell	25K	13.7%	48	55
bpr24	macro-cell	5K	14.7%	4	7

† on a Sparcstation-5

Table 1: *Relative Reduction of Spot Defects using Yield Optimization with Fixed Area. Total run times include compaction, simultaneous* WLM *and yield optimization, and IO.*

Simultaneous Yield and Wire-Length Optimization

Often yield must be optimized while simultaneously considering other constraints such as wire-length. For wire-length minimization the cost of each edge is linear in the edge length. Hence the wire-length cost is also a convex function. Since the sum of convex functions is convex, yield and wire-length can be simultaneously optimized. The yield cost on an edge usually dominates for small values of $\lambda = l(a \rightarrow b)$ and the wire-length cost dominates for large values of λ. Hence yield and wire-length cost interact minimally with each other and simultaneous optimization of yield and wire-length produces near optimal results for both yield and wire-length.

4 Experimental results

The algorithm described in this paper has been implemented in the C language and integrated in a high-performance 1-dimensional graph-based compaction system called ACP which is integrated into the Cadence DFII environment. The algorithm was tried out on several large industrial macro-cell and standard-cell style layouts, as well as on a few custom-style layouts, all implemented with sub-micron CMOS processes. Experimental data for the defect size distributions is used to evaluate the spot defect decrease after optimization. This experimental data was derived from actual industrial test cases.

Yield was optimized simultaneously with wire-length by adding a wire-length cost function to the yield cost function. Experiments using only WLM and no yield, and yield optimization with no WLM were also performed. Our results show that the wire-length is not significantly affected by simultaneously optimizing yield and wire-length. Also no significant benefits were observed for the yield when only WLM was performed.

Table 1 shows the result of applying the yield optimization algorithm with minimum area on a few industrial test cases. For each test case compaction is first performed and the probability of faults due to spot defects is computed before and after yield optimization using distribution (3). The *fault reduction* column in Table 1 shows the computed percentage of reduction in the number of faults due to spot defects. The size of the layout is the number of constraint graph vertices which is approximately the number of objects in the layout. The number of segments used in the piecewise linear approximation is about four. All our test cases show a significant improvement in yield even though area was maintained at its minimum size and the layouts were quite tight.

5 Conclusions

A novel compaction-based method for the optimization of yield is presented. By carefully spacing objects in the layout the probability of bridging faults due to spot defects is minimized. The problem of optimizing yield is reduced to a network flow problem, which is solved using efficient algorithms. Yield can be optimized with minimum area using the longest-path algorithm followed by the enhanced network flow algorithm. By increasing layout area if necessary, the algorithm described in section 3 can be used to maximize the number of good chips per wafer. Our method can, in theory, provide the largest possible reduction in the probability of bridging faults that can be achieved without changing the layout topology.

Preliminary results indicate that run-times are on the order of other compaction algorithms such as graph-building, Bellman-Ford and WLM. Hence the performance overhead for yield optimization is quite acceptable. Results run on industrial test cases simultaneously optimizing for yield and WLM with minimum area show a considerable reduction in the number of faults. Given the high economic impact of yield improvement in today's deep sub-micron fabs, these results show that our method can play a significant role in the reduction of manufacturing costs.

Test cases using the variable area approach are being evaluated. Other optimization objectives can be considered, as long as they can be effectively approximated with convex PWL functions of the constraint graph edge lengths. Wire capacitance has this property, and therefore objectives such as power, crosstalk noise and delay can be optimized using our method. Our current research is directed to the simultaneous optimization of several such objectives, using graph manipulation algorithms.

References

[1] A. J. Strojwas, "Design for Manufacturability and yield", in *Proc. IEEE/ACM DAC*, pp. 454–459, 1989.

[2] W. Maly, "Cost of Silicon Viewed from VLSI Design Perspective", in *Proc. IEEE/ACM DAC*, pp. 135–142, June 1994.

[3] A. Pitaksanonkul, S. Thanawastien, C. Lursinsap and J. A. Gandhi, "DTR: A Defect-Tolerant Routing Algorithm", in *Proc. IEEE/ACM DAC*, pp. 795–798, 1989.

[4] K. Balachandran et al., "A Yield Enhancing Router", in *Proc. ISCAS*, pp 1936–1939, June 1991.

[5] S. Y. Kuo, "YOR: A Yield-Optimizing Routing Algorithm by Minimizing Critical Areas and Vias", *IEEE Trans. on CAD*, pp. 1303–1311, September 1993.

[6] E. P. Huijbregts, H. Xue and J. A. G. Jess, "Routing for Reliable Manufacturing", *IEEE Trans. on Semic. Manuf.*, vol. 8:2, pp. 188–194, May 1995.

[7] V. K. R. Chiluvuri and I. Koren, "Layout-Synthesis Techniques for Yield Enhancement", *IEEE Trans. on Semic. Manuf.*, vol. 8:2, pp. 178–187, May 1995.

[8] R. Varadarajan and G. Lakhani, "A Wire Length Minimization Algorithm for Circuit Layout Compaction", in *Proc. ISCAS*, 1987.

[9] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1991.

[10] C. H. Stapper and R. J. Rosner, "Integrated Circuit Yield Management and Yield Analysis: Development and Implementation", *IEEE Trans. on Semic. Manuf.*, vol. 8:2, pp. 95–102, May 1995.

[11] C. H. Stapper, "Modeling of Defects in Integrated Circuit Photolithographic Patterns", *IBM J. of Res. and Dev.*, vol. 28:4, pp. 461–475, July 1984.