2–D Compaction
A Monte Carlo Method

Richard C. Mosteller
Computer Science Department
California Institute of Technology
Pasadena, Ca. 91125

Alexander H. Frey
IBM and Computer Science Department
California Institute of Technology
Pasadena, Ca. 91125

and Roberto Suaya
Schlumberger Palo Alto Research
3340 Hillview, Palo Alto, Ca. 94304

Abstract

In this paper we study the two-dimensional compaction of integrated circuit layouts. A curvilinear representation for circuit elements, specifically chosen to make the compaction efficient, is developed. A Monte Carlo algorithm with heuristic termination criteria was applied to a variety of designs. These experiments give running times for compaction that are consistent with a conjectured average complexity of $O(N^{3/2}\log^2(N))$ where N is the number of non-wire primitives in the cell. These experiments also produced favorable comparisons with hand-designs and with designs using iterated applications of one dimensional compactors. Several cells also were fabricated and tested to demonstrate the practicality of the representation and the compaction technique.

1 Introduction

Integrated circuit compaction reduces the area of a cell while preserving both the topology of the circuit layout and the Geometric Design Rules —GDRs. The Geometric Design Rules are constraints needed for manufacturing, associated with the relative position and overlap of the different components —electrically active devices and interconnecting wires— of the circuit.

Manufacturing considerations in VLSI provide a considerable incentive to minimize the silicon area required for the design of a cell since

$$\text{yield} \approx \exp(-\text{area}).$$

In this paper we address the problem of minimizing area, and describe a working system used for the automatic generation of near-optimal MOS layouts of cells. We use, as an optimization criterion, a function that combines

considerations of wire lengths and the distance of structures from a specified interior line.

Previous work in two dimensional compaction[11,9] has been limited exclusively to "Manhattan" geometry, *i.e.* rectangles oriented along cartesian axes. Hand designers do not limit themselves always to orthogonal geometry as shown in Figure 1 —a hand-designed nMOS cell. Tightly compacted cells can be produced this way. Curvilinear hand compaction has, to date, been an art that requires a highly skilled designer. It is, moreover, time consuming and error prone. But when density of the layout is a primary concern, as it is in custom VLSI, no automatic tool previously has been able to obtain a density comparable to that achieved by a craftsman artwork designer. One purpose of the work reported in this paper has been to break that barrier.

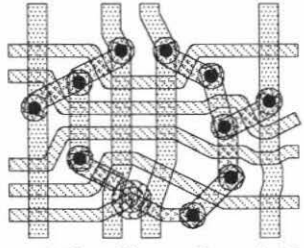

Figure 1 Curvilinear Sample Design

Issues like portability, scaling and updatibility can all be addressed in a simplified manner with this two-dimensional compactor —as can decisions related to composition such as port position and cell shape *e.g.*, aspect ratio for a rectangular bounding box.

We now will give a guided tour of the compaction process through a simple example. Consider a 4:1 multiplexor whose schematic transistor representation is shown in Figure 2. The input description is a loose geometrical specification of the cell shown in Figure 3. Notice how transistors are made out of the primitives, bubbles and wires, and how wires can bend around bubbles.

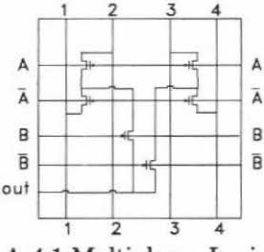

Figure 2 A 4:1 Multiplexor Logic Diagram

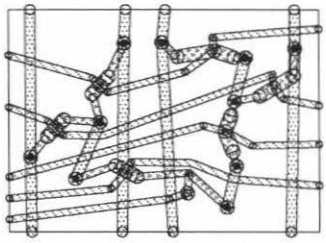

Figure 3 The 4:1 Multiplexor Initial Design Configuration

Guaranteeing the correctness of the input Geometric Design Rules is done by an interactive checker within the compactor's front-end editor. The designer's task is simplified by the allowed looseness of the input. The tedious, time consuming, and error prone phase of shrinking the cell to minimize its area —for a given pitch if necessary— while preserving the GDR's becomes a task for the compactor.

After the cell is entered, the compaction is performed using the simulated annealing technique and optimization schedule similar to that described by Kirkpatrick[3]. The annealing process optimizes a cost function made up of a monotonic function of the wire lengths plus a central potential acting on the bubbles. The cost function chosen reflects extensive experimental work[7]. Figure 4 shows the cell after several attempts to move all the bubbles at high temperature. During bubble motion, wires are bent and pushed around obstacles.

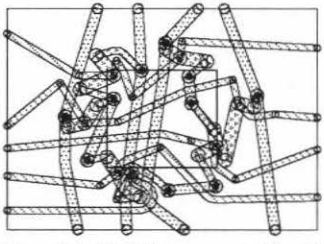

Figure 4 The 4:1 Multiplexor at Cooling stage 1

As the temperature is gradually lowered, the bubbles move slowly toward the center of the cell, as shown in Figure 5 and Figure 6. Although bubbles are pulled together somewhat along the paths of the wires, the major force pulling the bubbles towards the center of the cell is the central potential. At low temperatures, the central potential can produce some distortions on the layout, particularly around the edge of the cell. This distortion is removed by performing a final —minimum temperature— compaction after turning off the central potential. While doing this, a wall —an infinite potential barrier— is placed at the minimum bounding box of the cell. The resulting cell is compared with a hand layout in Figure 7. It is instructive to note that it took 0.5 CPU hours on a DEC-SYS-20/60 to compact this cell.

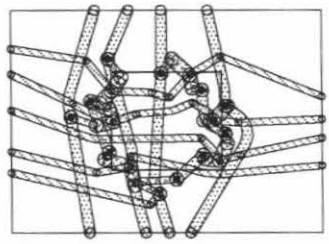

Figure 5 The 4:1 Multiplexor at Cooling stage 2

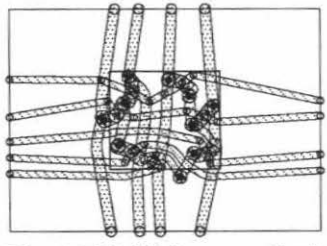

Figure 6 The 4:1 Multiplexor at Cooling stage 3

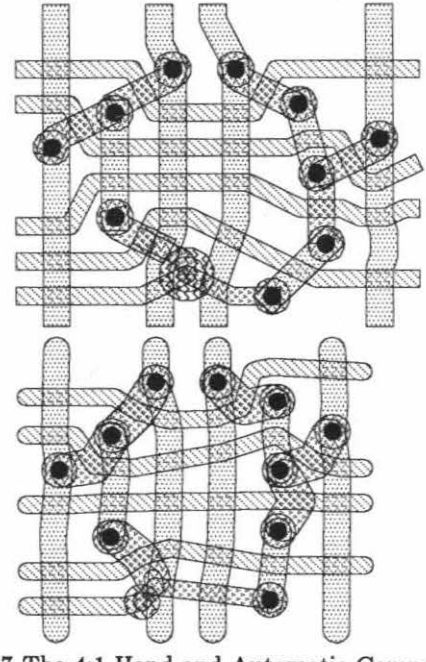

Figure 7 The 4:1 Hand and Automatic Compacted Cell

The "Bubbleman" implementation system was written, using the Mainsail programming language, by R. Mosteller[7]. Figure 8 shows a block diagram of the system.

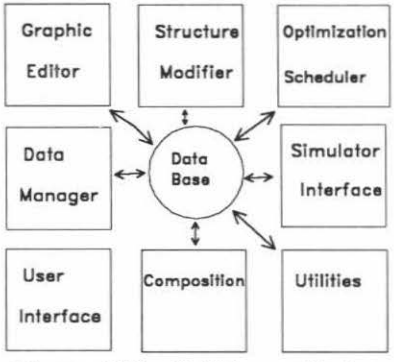

Figure 8 The Bubbleman System

It consists of 8 modules interacting with a data base. When circuits are initially entered –through the Interactive Graphic Editor– a check is made to assure that no GDR's are violated. The GDR's continue to be satisfied throughout the annealing process. A composition module is provided to allow a hierarchical approach making use of both the editor and the compactor functions.

In Section 2 we will discuss the representation. The primitive data types, *bubbles and wires*, are described together with how they are used for GDR checking. Composites of the primitives, called models, are used to build transistors and contacts. The special characteristics of these structures are briefly discussed.

In Section 3, the main body of the algorithm is presented. The *clear path* algorithm, the move construction, and the optimization strategy based on simulated annealing all are discussed.

Finally, in Section 4, we show some measurements of compaction performance on test cells. Some meaningful comparisons with hand layouts and with semi-automatic one-dimensional-compactor-generated cells are given. Some rough estimates from Section 3 of the growth of complexity with cell size are compared to limited test results. Finally, a simple adder cell was designed using this representation, compacted by this simulated annealing program, and successfully fabricated and functionally tested.

2 Representation

To minimize the computational cost of simulated annealing in two dimensional compaction, an easily-modified representation is required. Speed often

comes with simplicity, but the variety of objects —transistors, contacts, resistors, wires, etc.— and the variety of object sizes found in VLSI circuits makes finding such simplicity in a representation more difficult. For example, a new curvilinear representation developed by Whitney[10] was not designed for fast modification and update. Also, transistors and contacts in that representation were not designed to be malleable during compaction. Updating the data structure of EARL, a curvilinear representation developed by Kingsley[2], was also very time consuming.

2.1 A Simply-Modified Representation

Earlier applications of simulated annealing to molecular dynamics suggested a new representation. Only two primitive components are used. Circular objects called *bubbles* are connected by *wires* as shown in Figure 9. Bubbles are hard objects that will be moved around as if they were molecules during the annealing process. Wires connecting bubbles are stretchable objects which are continuously modified to follow the shortest path between the bubbles they connect while preserving all required Geometric Design Rule constraints. The wires act as an elastic medium between the bubbles they connect during the annealing process.

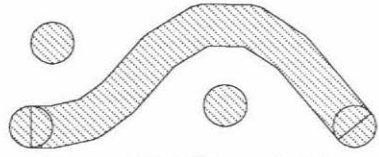

Figure 9 Bubbles and Wire

In this representation bubbles are solid circular objects that cannot be distorted or moved by other objects. Wires are pliable but are always stretched tight so as to follow the shortest path between bubbles that preserves the initial topology of the circuit without violating any GDR's. Each wire consists of an alternating sequence of pieces called segments and arcs, always starting and ending with a segment. A segment may have zero length.

VLSI circuits are designed and fabricated in several layers. Structures such as contacts or transistors are composed of bubbles and wires with several layers per primitive. These structures form models for each desired type and size of such structures, and their *names* are included in the menu of the graphic editor so that they can be instantiated as needed by the designer. After a model is instantiated, the individual objects in the structure are treated independently by the move algorithm. This enables the shape and orientation of the structure to be modified during the compaction process, to make better use of the total space available.

To achieve a compact representation, all primitive objects are divided into types, where the characteristics of each type are defined by a *rule* for

that type of object. There are very many objects of relatively few types in any large cell. Thus the unique data associated with the object can be limited to position and connectivity information plus a rule reference, and all the other information needed for GDR checking can be placed in the rule definition.

The GDR's for the several layers of design and fabrication are described through a set of tables reflecting the technology of the current implementation. In the tables describing the technology of the design are minimum separation values, MinS_L, for any two objects in a layer. The rule associated with an object, O_i, not only indicates when the object is in the layer, L, but also specifies, W_i, the width of that wire or the diameter of that bubble in that layer. Using these values, we can compute the minimum allowable separation distance, $\mathrm{MinD}(O_1, O_2)$, for two objects, O_1 and O_2, in the layer,

$$\mathrm{MinD}(O_1, O_2) = \frac{W_1}{2} + \frac{W_2}{2} + \mathrm{MinS}_L.$$

To check that O_1 and O_2 do not violate a GDR in L, the distance between the closest points of O_1 and O_2 is computed and compared to $\mathrm{MinD}(O_1, O_2)$.

For the purpose of efficient GDR checking, each rule contains several arrays of Boolean variables. These arrays are used to determine what other types of objects must be kept away from this type. Speed is obtained by using only logical functions for determining this. Two objects are seen to be in the same layer when the Boolean variable corresponding to that layer is set to 1 in the rule for each object. For example, the diffusion layer is called *green*. A value, 1, for the green variable in the rule for an object indicates that that type of object has content in the diffusion layer. Two objects with green content are detected by ANDing the array of color variables in each of their rules. The technology file is referenced to see how far the green content of these two objects must be kept separated.

Wherever a wire is connected to a bubble we impose a constraint that the bubble diameter must be equal to or greater than the width of the wire. Not only does this avoid any sharp corners, but it greatly simplifies GDR checking. In particular, the end of a segment does not have to be considered separately, even when the direction that a wire leaves a bubble is rotated.

When there is a design rule between two different fabrication layers, a new hypothetical layer is created to implement this rule. For example, to implement a design rule between polysilicon —red— and diffusion —green—, a hypothetical color *red/green* is created and all objects with either red or green content also are given red/green content. Thus a red object and a green object are found to have the red/green layer in common and must be kept separated by the distance indicated for red/green in the technology file.

Provision is also made whereby one color may supersede another by a logical check of additional Boolean arrays in the rules for two objects. This provision might need to be used, for example, if the red/green separation distance were greater than the red separation distance. Two red objects would

have both red content and red/green content. In this case the red/green distance could be ignored and only the red distance applied.

There are some cases where two objects that would normally be separated by a GDR should be treated differently. For example, if two objects are electrically connected, they are not subject to the same GDR as they would be if they were not connected. A concept called *related* is used to handle these exceptions. Electrical connectivity is the most common reason for two objects to be related, although there are other reasons, in particular, for objects in a model. Each primitive object has a *related number* associated with it that can be compared with the related number of another object to determine when to apply the *related arrays* in their respective rules for computing the required separation distance. For most objects not in models the related number corresponds to a net number.

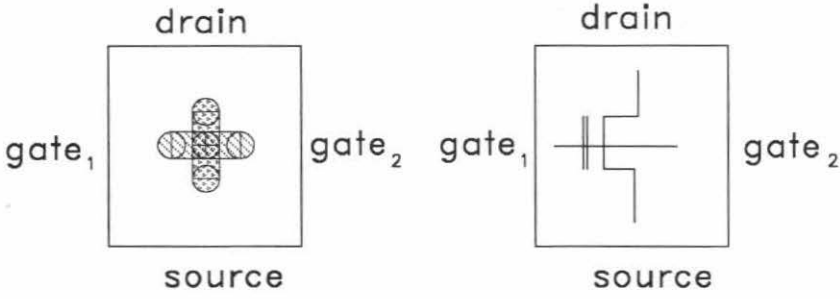

Figure 10 Enhancement Transistor Model Example

The identity of each object in the model is continually maintained in the data structure. By carefully choosing the rules between particular objects in a model, excessive distortions are allowed to occur at high temperatures during annealing, and are eventually removed as the total circuit is cooled. For example, consider the model for a minimum size nMOS enhancement-mode transistor together with its circuit representation, as shown in Figure 10. Observe that the model contains five interconnected bubbles. The need for several bubbles is illustrated in Figure 11 that displays an example of excessive distortion in such a transistor. Figure 12 shows how a strong attraction between the four outside bubbles and the center bubble insures that the electrical characteristics of the transistor measured by length and width are preserved in the final compacted cell. Similarly, invalid and valid buried contacts are shown in Figures 13 and 14.

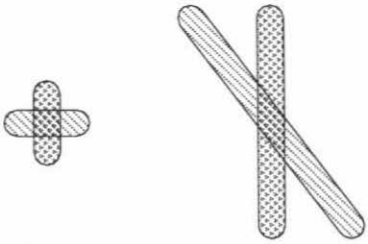

Figure 11 Invalid Enhancement Transistor

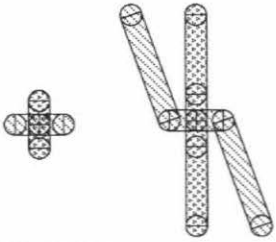

Figure 12 Valid Enhancement Transistor

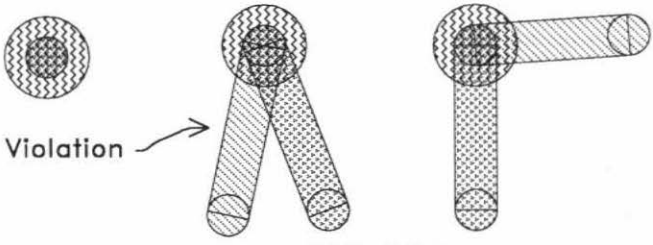

Figure 13 Invalid Buried Contact

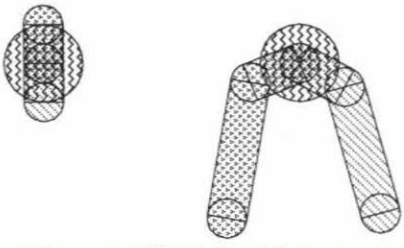

Figure 14 Valid Buried Contact

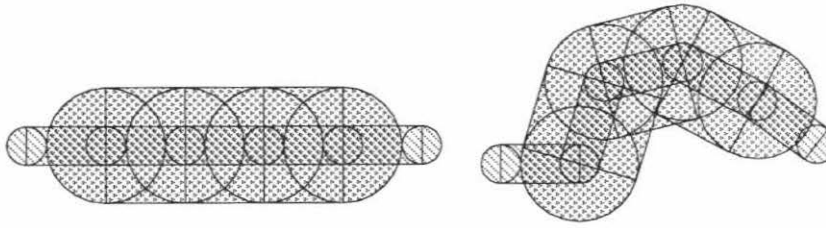

Figure 15 Snaky Enhancement Transistor

Wide transistors, as shown in Figure 15, illustrate another application of hypothetical colors in the design of models. The distance required between adjacent bubbles is determined by the rule specified for each bubble. By including two hypothetical colors alternately on the bubbles, the distance between these colors limits how close the alternate bubbles can get. When the adjacent bubbles are tightly packed, this distance between alternate bubbles determines the angle of bends in the transistor. For example, a minimum separation between alternate bubbles of $\sqrt{2}$ times the minimum separation of adjacent bubbles will limit the angle of bend to 90 degrees after cooling. Of course, if the designer does not want to allow any bends in the wide transistor, no intermediate bubbles are needed and the width is determined by the separation of the two end gate bubbles. The transistor will be straight when cooled if the pull of the gate wire on the end bubbles is large enough.

The concept of related objects being exceptions to the standard GDR's is particularly useful in the design of models. For example, in the minimum-size enhancement transistor shown above in Figure 12, the two diffusion bubbles on opposite sides of the gate are not electrically connected. However their minimum separation distance should be 2λ rather than the usual 3λ distance for unrelated diffusion primitives. Their separation distance from their transistor gate and polysilicon is also less than from the gate and polysilicon of other unrelated transistors. To provide for all these relations, primitives in models are given four related numbers compared to only one for non-model primitives.

3 Algorithms

We will compact our design to a smaller or different shape in a sequence of small steps to produce large modifications. Each small step is a move of a single bubble in a straight line to a new position where the following invariant conditions are preserved; the Geometric Design Rules are satisfied at all times, the topology of the initial design is maintained, and all wires connecting bubbles follow the shortest path that preserves that topology.

3.1 The MOVE Algorithm

A bubble move is an atomic action —that is, only one bubble may be processed at a time. We do not allow one bubble to push or move another bubble in this process. The fact that all GDR's have been represented using single layer constraints, allows the move process to consider each primitive in each layer independently. To see that the proposed position of two primitives does not violate any GDRs, it is sufficient to see that they are not violated in any single layer. Therefore the distance between two primitives must be at least the maximum of the single layer requirements. We will now describe how the move processing works in a single layer. An example of a cell before and after a bubble move is shown in Figure 16.

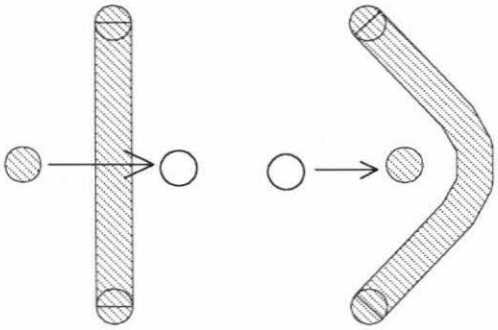

Figure 16 Bubble Move Example

A complex move example, one that has been carefully chosen to contain most of the special cases a move process must consider, is shown in Figure 17.

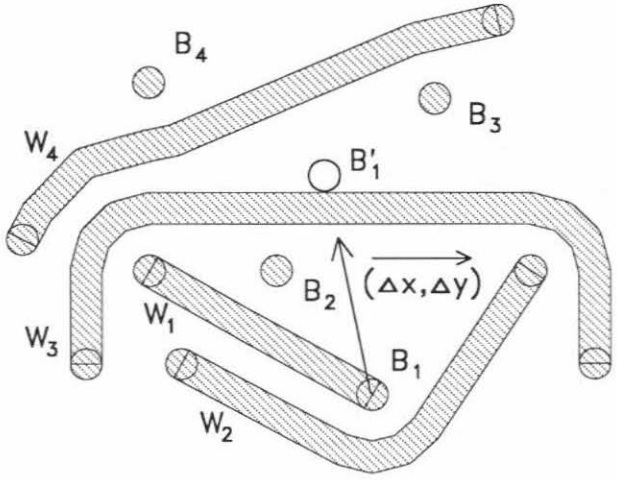

Figure 17 Bubble Move Example, Before Move

The example shows some of the interesting deformations to wires that are needed in order to preserve the invariant conditions. Notice that the bubble B_1 in Figure 17 has an attached wire W_1 which will need to wrap bubble B_2 in the new position. The arc of wire W_2 which wraps bubble B_1 may need to be unwrapped, and in turn, W_2 may need to wrap the arc of wire W_1 around B_2. Now look at wire W_3. It will be pushed by bubble B_1 and thus will need to be wrapped around it. In a similar manner wire W_4 will need to be wrapped around that arc of W_3. There is also an arc of W_4 around B_3 that may possibly need to be unwrapped.

Before we move the bubble B_1 to its new position, we need to check that the bubble can be moved directly —*i.e.* in a straight line— to that position without violating any GDRs and without moving any other bubbles on the way. If we can move the bubble without causing any design rule errors along the way, then the move may be performed. The move process is thus divided into two parts. The first part is the *clear path* algorithm which determines whether the bubble can be moved with no errors along the desired straight-line path. The second part is the *queue construction* which builds the required add and delete queues of primitives to implement the actual move. The area that must be free of bubbles in the above example —that is, the required clear path— is shown in Figure 18.

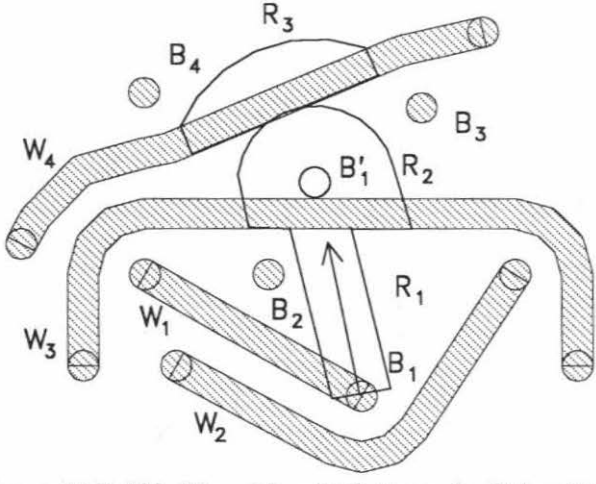

Figure 18 Bubble Move Clear Path Example, Before Move

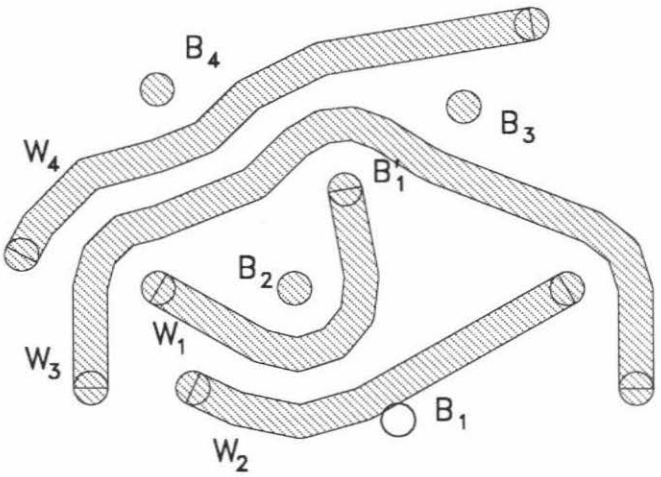

Figure 19 Bubble Move Example, After Move

The position of all the bubbles and wires after the move is shown in Figure 19. In constructing the add and delete queues to implement the move, the order in which the wires are considered is important. The wire furthest in the direction that the bubble is being moved should be considered first, then the next furthest. The last wire considered should be the wire furthest in opposite direction from the bubble move.

Notice that, in both clear path construction and queue construction, one primitive is considered at a time. In the case of a multi-layer move, all the layers are considered for each primitive before going on to the next primitive. This is all that is required to go from the single-layer case to the multi-layer case.

3.2 MOVE Algorithm Run Time

Both checking the clear path for a move and constructing the add and delete queues —*i.e.* moving relevant wires— require searching areas of the cell for primitives. The clear path area is defined by the bubble move and the wires being pushed by that bubble move. For queue construction, each wire move requires searching one or more triangles across which the wire would move if there were no bubbles in the way. Each triangle side may end in an arc which complicates the triangle search process. A tree search is used so that the search time only grows with the logarithm of the number of primitives in the cell. Part of constructing the add and delete queues for each move is updating the search trees for each primitive that is modified.

The time for a move depends on two factors, the time to search and update the search tree, and the density of bubbles and wires around the bubble being moved. The first grows with the logarithm of the number of

bubbles in the cell. The second is a function of the compactness of the cell. The central potential may cause large cells to be more compact on the average in the interior than small cells. But this increase in density is expected to be only a small factor in the increase of CPU time for a bubble move. Experimental data is consistent with a very slow growth of the average time for a single move during a compaction process —see Section 4 where the increasing density factor appears to be bounded by $\log(N)$, giving an overall average move time growth of less than $\log^2(N)$.

3.3 The Compaction Algorithm

Two dimensional compaction is known to be NP hard[8]. This does not preclude the existence of algorithms that can obtain reasonable near-optimum configurations in polynomial time. A careful use of the Monte Carlo method known as simulated annealing[3] provides a heuristic optimization algorithm.

This optimization method uses a *cost function* for each configuration in the domain of optimization. A key to success when using simulated annealing is the choice of this cost function for each configuration. Near-minimums of the cost function must correspond to near-optimum configurations, and when a configuration is modified, the corresponding delta to the cost function should be easily computable from information local to the modification.

The cost function we have chosen has two major parts: 1) a wire cost, based on the length of each wire, to keep wires short, and, 2) a central potential cost, based on the position of each bubble and arc, to pull the various primitives toward the center of the cell.

A normal wire contributes a cost proportional to its length. A coefficient of cost is chosen for each layer, larger coefficients being associated with the less desirable layers, *e.g.*, diffusion or polysilicon. For wires that are part of models, *e.g.*, transistors, buried contacts, etc., the cost of each wire is a quadratic function of its length. This allows the models to move without spreading out too far, and delta wire costs still can be calculated with only local information.

The central potential is implemented in several pieces. A cell wall is given, beyond which there is an infinite potential. A second boundary, called a membrane, is used to control a quadratic growth of the central potential cost inside the wall. The shape of the membrane is chosen to approximate the desired form factor of the compacted cell. Outside of the membrane, the central potential cost is a quadratic function of the distance from the membrane. For bubbles inside the membrane, the central potential cost is another quadratic function of the distance of the bubble from the center of the cell. The quadratic central potential costs are then chosen so that the derivative of the central potential cost for a bubble located at the membrane is roughly equivalent to the derivative of the cost for a typical single wire attached to the bubble. Port bubbles are only affected by the central potential parallel to their boundary.

The resulting form of the cost function for any configuration is

$$H = \sum_{n=1}^{N_w} C_w(W_n) + \sum_{j=1}^{N_{mw}} C_i(W_j) + \sum_{k=1}^{N_b} C_{mb}(B_k) + \sum_{l=1}^{N_a} C_{ma}(A_l) + \sum_{m=1}^{N_p} C_p(P_m),$$

where n indexes the non-model wires with costs C_w, j indexes the model wires with costs C_i, k indexes the bubbles with C_{mb} being the central potential cost function for bubbles, l indexes the arcs with C_{ma} being the central potential function for arcs, and m indexes the ports with central potential cost C_p.

When any move is considered in the compaction process, only the delta cost is computed, *i.e.* the difference in the contribution to the cost function for each primitive that is changed by the move. The rule to determine if a considered move should be implemented is as follows:

1. The move is rejected unless it can be implemented without violating any geometric design rules.
2. If it does not violate any geometric design rules and if the delta cost of the move is negative, then the move is implemented.
3. If it does not violate any geometric design rules and the delta cost of the move is positive, then the move is implement if and only if
$$R \leq e^{-\Delta H/T}$$
where T is the temperature, and for each move, R is chosen to be an independent uniformly distributed random variable between zero and one.

The move length is decreased monotonically from a value equal to the average bubble size at high temperature to a minimum of 0.25λ at low temperatures. The move size is large at high temperatures so that objects can move more quickly to configurations in equilibrium. As the temperature is lowered, the cell becomes more compact, and the move size is decreased to avoid too many move rejections caused by GDR violations. When the move size is changed, the temperature should also be lowered. Ideally one would try to continually balance the two independent parameters, temperature and move size, to have a similar average probability of acceptance —for moves that do not violate any GDR's— whenever the move size is changed. In practice, the decreasing move size was implemented with two move sizes, average bubble size and 0.25λ. This was found to be satisfactory.

With regards to the Monte Carlo simulation strategy, bubbles are chosen randomly. When a bubble move is rejected by the clear path algorithm, another attempt to move the same bubble is made in a different direction chosen at random. If necessary, there may be up to four clear path rejections on a single bubble before a different bubble is considered. This strategy is slightly different than the strict Metropolis algorithm[4]. The convergence

properties have been shown experimentally[7] to be slightly better than the Metropolis strategy.

We have chosen an exponential temperature cooling schedule $T_n = 0.8 * T_{n-1}$. The initial temperature is chosen to be

$$T_0 = \frac{W_{max}}{-\ln(P)} + \frac{C_{mem}}{-\ln(P)},$$

where W_{max} is the maximum wire cost, C_{mem} is the cost of a one step move for a bubble when the bubble is at the cell boundary. The normalizing probability, P, is chosen to be 0.5. The process of compaction occurs in three stages. The first stage is characterized by high temperatures and large move sizes. The second stage uses lower temperatures and small move sizes. The final stage is used to reduce the distortion due to the central potential. It eliminates the central potential and uses temperatures gradually approaching freezing with small move sizes.

The schedule is independent of the number of bubbles and is the same for all cells. Bubble moves are grouped into passes, where a pass consists of an attempt to move N bubbles, N being the number of bubbles in the cell being compacted. At each temperature the process runs for a variable number of passes until a rough measure of equilibrium is reached. We have used a heuristic criteria to indicate when equilibrium has been reached. We continually compute the average delta cost over the last k passes. When this function changes sign then equilibrium is assumed. In most experiments, we have let $k = 8$.

3.4 Compaction Algorithm Run Time

The run time complexity of the compaction algorithm itself is very difficult to estimate. Some parameters other than cell size whose adjustment is important to the compaction CPU time are membrane position, move length, annealing schedule, and equilibrium stopping criteria. In larger cells, primitive elements have further to move on the average between their initial position and their final position. We do not know that this time should be linear with distance. But if it were, then the time —measured in required number of bubble moves— would be increasing linearly with the square root of the number of bubbles. We use this as a first order estimate of the growth of the number of passes.

Combining this factor with the earlier estimates of bubble move time and density factor, we then estimate the complexity of the algorithm to be approximately $O(N^{3/2}\log^p(N))$ with $p \leq 2$. Of course, the search for the optimal configuration will require exponential time. The algorithm does not guarantee convergence to anything but a local minimum. In practice, the algorithm does perform very well, —roughly an order of magnitude faster on a one Mflop machine than a very good designer to achieve smaller final areas.

4 Experimental Results and Analysis

This section describes a subset of the experiments performed using the Monte Carlo compactor. Additional experiments chosen to analyze and optimize the parameters in the cost function are described elsewhere[7]. The next subsection gives some results indicating the run time sensitivity to membrane position. After that, some rough measures of average complexity of the compaction method are given using a controlled set of experiments for a range of cell sizes. Some meaningful comparisons with hand designs are shown. To show that the curvilinear framework can build working designs, a chip consisting of three cells was fabricated and tested.

4.1 Choosing Membrane Position

The position of the membrane determines the strength of the central potential. To gain an idea of the effects of the central potential, we will show three versions of a compacted cell, one without a central potential, one with an overzealous central potential, and one with a normal central potential. Consider the uncompacted cell shown in Figure 20. The cell, compacted without a central potential, is shown in Figure 21. It is interesting to observe that this cell is not very compact although the wires have been pulled to their shortest lengths.

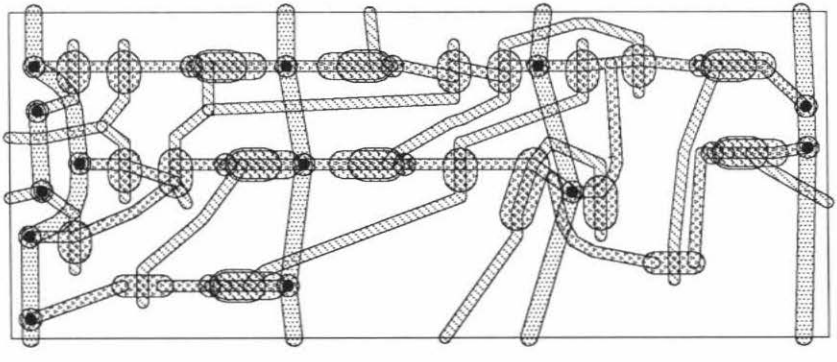

Figure 20 Input Cell

Now consider what happens when the central potential cost is too high —see Figure 22. In this case, the transistors are deformed, and the wires are overstretched by the bubbles pulled to the center. An interesting point about the overzealous membrane is that it causes the running time to go up.

Finally a compacted version with a suitable central potential is shown in Figure 23. Notice that the transistors are well formed, and the wires are not overly stretched.

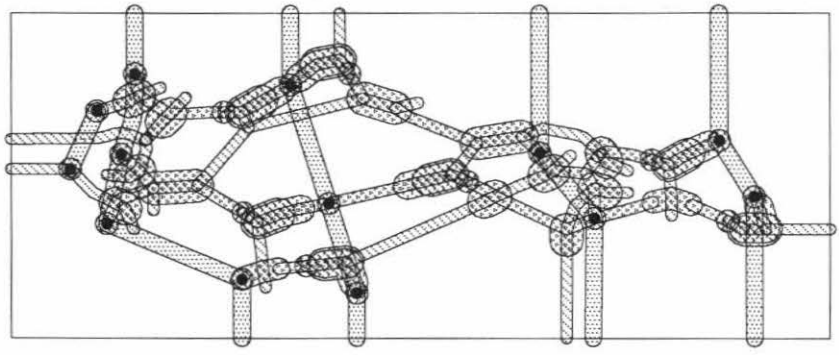

Figure 21 Compaction without central potential

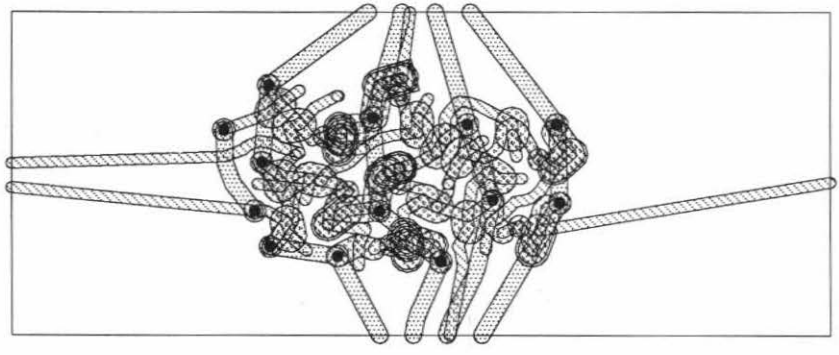

Figure 22 Overzealous central potential

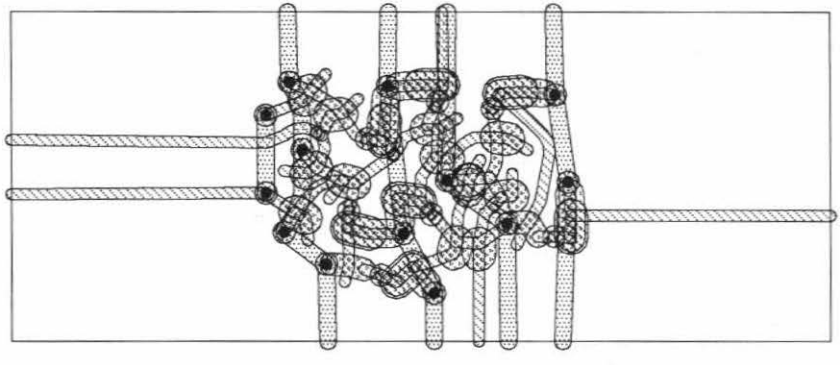

Figure 23 Suitable central potential

A properly positioned membrane will pull the bubbles inside the membrane without undue distortion. The membrane perimeter is intended to enclose the region of the final compacted cell and thus describe the final desired shape.

An experiment was run to test the effect of the membrane position on both the run time and the effectiveness of this compaction algorithm. A test cell was compacted with the membrane set at a computed *ideal* [1] area. Then the same cell was compacted again with the membrane dimensions increased by a factor of two, then again with the membrane dimensions increased by a factor of four, and again with the membrane dimensions increased by a factor of six. The results are shown in Table 1. This indicates that positioning the membrane at roughly two times the *ideal* minimum area dimensions gives a good balance between a low CPU running time and good cell compaction.

Membrane scale	Cpu time	Compact size	Actual/Ideal
1	6:0:8	$5451\lambda^2$ $79\lambda \times 69\lambda$	1.76
2	4:49:17	$5440\lambda^2$ $80\lambda \times 68\lambda$	1.73
4	3:42:19	$5762\lambda^2$ $86\lambda \times 67\lambda$	1.86
6	4:40:58	$6390\lambda^2$ $86\lambda \times 67\lambda$	2.07

Table 1 Membrane Size Statistics

4.2 Run Time Experiments

To evaluate our estimates of how the complexity grows with the number of bubbles, we have run a set of timing tests using larger and larger cells. The base cell is the multiplexer cell shown earlier in Figure 3. To form a sequence of increasingly larger cells, we combine this cell with itself. By constructing the sequence in this way, we keep an approximately uniform topology across the test set. The sensitivity of the run time to topology is not clear. However, other experiments[7] have indicated that the dependence is quite small.

The base cell has 60 bubbles. When two of the base cells are combined to form Cell 2, the number of bubbles is not twice the bubbles in test cell 1. The bubbles that are along the seam of composition are absorbed, thus slightly reducing the total bubble count.

[1] The sum of the areas of the primitives plus the GDR regions around them, minus a detailed estimate of the allowable overlaps.

Each cell uses the same annealing schedule. The membrane for each cell is set at double the dimensions of the *ideal* minimum bounding rectangle. The central potential, determined by this membrane position, is the most critical parameter affecting the run time of this algorithm. The total running time for each of the test cells is shown in Figure 24 with processor time in hours plotted versus bubble count. The run time for each cell is shown by a $\times$. The continuous line is a plot of $N^{3/2}\log^2(N)$, with a multiplying constant chosen to make the line coincide with the run time of the 300 bubble cell.

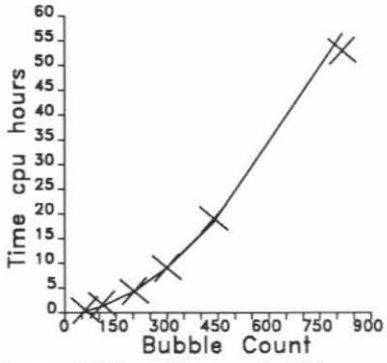

Figure 24 Run time vs. bubble count

To evaluate the complexity estimates for average move time and for time to reach equilibrium, we collected several statistics during these experiments. To estimate the growth of move time separately from density considerations, we looked at the final temperature of Stage 3 after the central potential was removed. The average move time for this temperature is plotted as a function of bubble count in Figure 25. The solid line shows the depth of the search tree with a multiplying constant chosen to make the line coincide with the time for the 300 bubble cell. By looking at these cells without the central potential, we hoped to minimize the effects of higher density in the larger cells.

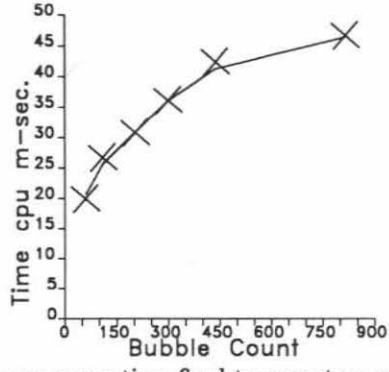

Figure 25 Average move time final temperature vs. bubble count

To measure the average move time including the density factor, we computed the average move time over the entire compaction algorithm. This time is plotted in Figure 26. In this plot, we included the line, $\log^2(N)$, with a multiplying constant chosen to make the line coincide with the time of the 438 bubble cell for comparison. This appears to bound the time growth. Thus, for this set of cells, the density appears to contribute no more than another factor of $\log(N)$.

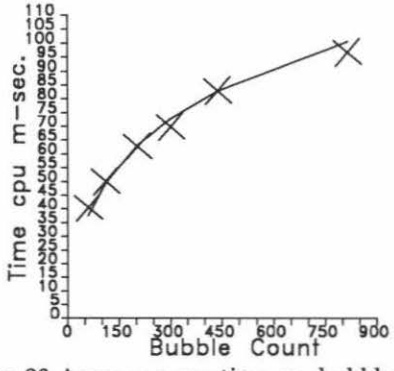

Figure 26 Average move time vs. bubble count

Finally, to estimate the growth in the number of bubble moves in the compaction process, we computed the total number of passes[2], minus the minimum number of passes needed to indicate equilibrium at a single temperature times the number of temperatures used in the entire compaction process. The resulting value is an estimate of the number of passes needed to reach equilibrium summed over all temperatures. This value is plotted in Figure 27 against bubble count. For reference, $kN^{1/2}$ is also plotted with a multiplying constant k chosen to make the line coincide with the run time of the 438 bubble cell for comparison.

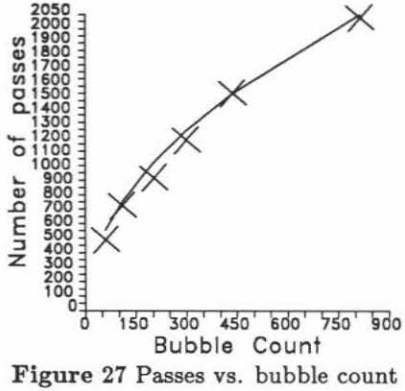

Figure 27 Passes vs. bubble count

[2] Recall that a pass is the attempt to move all N bubbles.

As discussed in Section 3.4, this gives a rough indication of how close to a linear function of $N^{1/2}$ is the time required for a bubble to move a given distance.

4.3 Comparisons With Other Methods

The cells for this experiment were chosen for their diversity. The first two cells shown in Table 2 are hand designed curvilinear cells, designed by Don Speck at Caltech. The next four cells were taken from a quanternary multiplier chip design[1], and were compacted using the Rest[6] one dimensional compactor system with human assistance. These last four cells used Manhattan geometry. An average of over a weeks worth of designer's time was spent compacting each of the six cells.

Cell	Bubbles	Hand Size	Compacted Size	Actual/Ideal
Mux1	60	$1376\lambda^2$ $43\lambda \times 32\lambda$	$1216\lambda^2$ $38\lambda \times 32\lambda$	1.34
Mux4	200	$5893\lambda^2$ $83\lambda \times 71\lambda$	$5396\lambda^2$ $76\lambda \times 71\lambda$	1.63
C2	119	5891λ $137\lambda \times 43\lambda$	$4760\lambda^2$ $40\lambda \times 119\lambda$	1.68
Cg	174	$5447.75\lambda^2$ $141.5\lambda \times 38.5\lambda$	$4340\lambda^2$ $35\lambda \times 124\lambda$	2.09
Bi1	307	$14490\lambda^2$ $140\lambda \times 103.5\lambda$	$12093.75\lambda^2$ $93.75\lambda \times 129\lambda$	1.88
Stuff	302	$16880\lambda^2$ $105.5\lambda \times 160\lambda$	$14000\lambda^2$ $100\lambda \times 140\lambda$	2.26

Table 2 Compacted Cells Statistics

Notice that in all cases the automatically compacted cells are smaller than the hand compacted cells. An important point to notice about the cell figures is that the cells could be compacted still further if some topological changes were introduced. Much of the time spent on the original design of these four cells was spent finding a topology suited to their Manhattan environment.

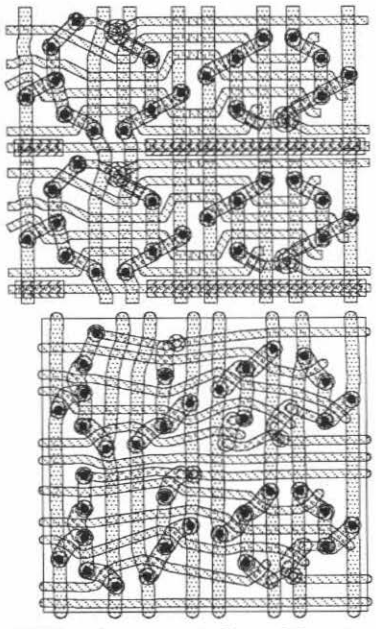

Figure 28 Hand compacted vs. Mux 4 annealed

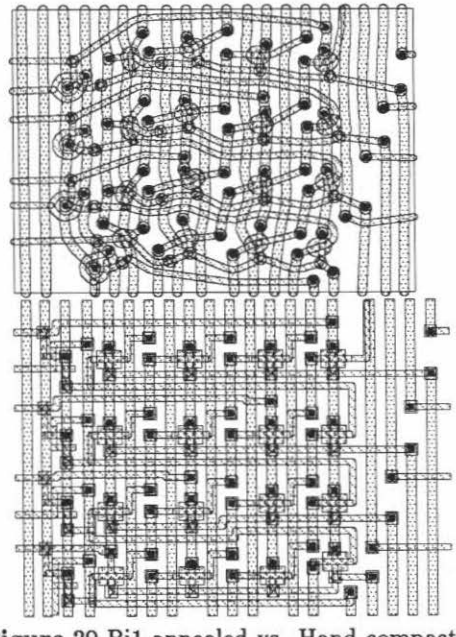

Figure 29 Bi1 annealed vs. Hand compacted

4.4 Fabrication Experiment

A one-bit adder was designed using the representation described in Section 2. Two topological instantiations of this design were compacted using the Monte Carlo algorithm. One of these was stacked four times to make a four bit adder and compacted again. These three cells were fabricated by MOSIS[5]. Four chips were returned and functionally tested. All of them worked perfectly.

5 Conclusions

In summary, we perform two-dimensional compaction on curvilinear geometry to obtain a compacted layout that is independent of the initial entry. The dependence on the topology is of course still there, but no other dependence on the initial geometry should be left. As we have seen this requires full two dimensional compaction plus automatic bending of wires. This is achieved in our method by treating the wires as rubber bands that bend around circular objects. Our desire has been to achieve layouts that are comparable in density to human layouts. This stems from the exponential dependence of the yield on the area. We also would like to have some control on the wire lengths. This is achieved through the cost function —higher costs produce shorter wires— as discussed in Section 3. We also wanted to have portability to other manufacturing technologies and scalability to other lithographies. By modifying the tables describing the technologies and rules, all of these features are automatically obtained with the Monte Carlo algorithm described, provided that no topological changes are needed.

Only rough estimates of the CPU requirements and the growth of complexity with cell size have been obtained. Including density variations, time for a single move appears to grow on the order of $O(\log^2(N))$ where N is the number of primitives in a cell. An overall complexity growth estimate of $O(N^{3/2}\log^2(N))$ seems consistent with experiments using a small number of cells.

References

[1] A. Frey.
Rabbit Chip.
California Institute of Technology, Pasadena California, 1983.

[2] C. Kingsley.
"Earl: An Integrated Circuit Design Language".
Masters Dissertation, California Institute of Technology, Computer Science Department, Technical Report #5021, Pasadena California,1981.

[3] S. Kirkpatrick, C.D. Gelatt, Jr., M.P. Vecchi.
"Optimization by Simulated Annealing".
SCIENCE, Volume 220, May 13, 1983, pages 671–680.

[4] Nicholas Metropolis, Arianna W. Rosebluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller.
"Equation of State Calculations by Fast Computing Machines".
The Journal of Chemical Physics, Volume 21, Number 6, June 1953, pages 1087–1092

[5] The MOSIS Project.
The MOSIS System(what it is and how to use it).
USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, California 90292-6695, March 1984.

[6] R.C. Mosteller.
"A Leaf Cell Design System".
Masters Dissertation, California Institute of Technology, Computer Science Department, Technical Report #4317, Pasadena California, 1981.

[7] R.C. Mosteller.
Monte Carlo Methods for 2–D Compaction.
PhD Dissertation, California Institute of Technology, Pasadena California, 1986.

[8] Sarma Sastry and Alice Parker.
"The Complexity of Two–Dimensional Compaction of VLSI Layouts".
IEEE International Conference on Circuits and Computers, ICCC 82, September 1982.

[9] Hiroyuki Watanabe.
IC Layout Generation and Compaction Using Mathematical Optimization.
PhD Dissertation, Computer Science, The University of Rochester, New York 1984.

[10] T.E. Whitney.
Hierarchical Composition Of VLSI Circuits.
PhD Dissertation, California Institute of Technology, Pasadena California, 1985.

[11] M. Schlag,Y.Z. Liao and C.K. Wong.
"An Algorithm for optimal two-dimensional compaction of VLSI layouts".
North Holland INTEGRATION, the VLSI journal 1, 1983, pages 179–209.