A Faster Compaction Algorithm with Automatic Jog Insertion *

Kurt Mehlhorn and Stefan Näher

FB10, Informatik
Universität des Saarlandes
6600 Saarbrücken

15/1987

Abstract: In this paper we refine work of Maley [13] on one-dimensional compaction with automatic jog insertion. More precisely, we give an algorithm with running time $O((n^2 + k) \log n)$, where $k = O(n^3)$ is a quantity which measures the difference between the input and the output sketch, and so improve upon Maley's $O(n^4)$ algorithm. The compaction algorithm takes as input a layout sketch; the wires in a layout sketch are flexible and only indicate the topology of the layout. The compacter minimizes the horizontal width of the layout whilst maintaining its routability. The exact geometry of the wires is filled in by a router after compaction.

* Research supported by DFG under contract SFB 124, TP B2

0. Introduction

A *compactor* takes as input a VLSI-layout and produces as output an equivalent layout of smaller area. An effective compaction system frees the designer from the details of the design rules and hence increases his productivity and on the other hand produces high quality layouts. For these reasons, compaction algorithms have gained widespread attention in the VLSI-literature [4,7,9,10,13,22] and are the basis for several computer-aided circuit design systems [3,4,11,19,21].

In this paper we restrict ourselves to one-dimensional compaction where the area is reduced by moving the features of the layout in a single direction only. For convenience, we assume this direction to be horizontal. For an approach to two-dimensional compaction we refer the reader to [7].

For the purpose of compaction a layout is separated into modules, which are rigid in shape and size, and wires, which are flexible. Most compaction algorithms use this flexibility in a very limited way. For example, the compactor in HILL [11,19] treats vertical wire segments as rigid objects during horizontal compaction and only changes the length of horizontal wire segments. Some compaction algorithms allow vertical wires to bend during compaction by the insertion of jogs. This is either done interactively by the designer ([4,21]) or automatically [20,21,22]. The latter procedures are however ad hoc and not guaranteed to be effective.

A major step forward was made by Maley [13] by putting the recent advances in homotopic routing to use in compaction. A *homotopic router* takes a layout sketch, consisting of the exact placement of the modules and the topology of the wires (cf. Figure 2), as its input and produces as output a (detailed) routing of the sketch in some wiring model (cf. Figures 3,4 and 5). For several different wiring models (one-layer routing in grids [8,14], gridless one-layer routing [5,14], knock-knee mode routing [6]) it is known that efficient homotopic routers exists. More precisely, it was shown that in these cases a simple cut condition, henceforth called routability condition, is sufficient for the routability of the sketch.

Maley [13] proposed to view, at least for the purposes of compaction, the wires in a VLSI layout, only as indicators of the layout topology, and to compact the layout maintaining the routability condition. The wires are constructed in their final form by a homotopic router. In this way, he puts automatic jog insertion on a sound theoretical basis. Maley shows that the compaction problem can be solved in time $O(n^4)$, where n is the size of the sketch. In this paper, we improve the running time to $O((n^2+k)\log n)$ where k is a quantity which measures how much the input and output sketch differ. We expect $k = O(n^2)$ in practice and always have $k = O(n^3)$. Our algorithm is not only faster than Maley's but also easier to understand. We want to emphasize however, that our correctness proof rests completely on the foundations laid by Maley.

The intuition behind our algorithm is quite simple. We put the input sketch between two rigid vertical bars and then move the right bar to the left. Initially, all modules except the right bar stay at their initial position. At some point, a tight cut (density = capacity) between a module and the right bar will arise. Therefore, this module starts

moving together with the bar. At some later point, some other cut becomes tight, either between a module and the right bar or between a module and the module which moves already with the bar. So a second module starts to move with the bar. We continue in this fashion until the left bar starts to move. At this point, we have computed a configuration of minimal x-width.

The simplicity of our approach makes it very flexible. In particular, we can also handle maximum distance constraints, we may compact in arbitrary directions and not just in x-direction and we can support plowing. A *plow* is a line segment which intersects no feature but apart from that can be in arbitrary position. A plow operation consists of moving the plow in a certain direction for a certain distance, cf. Figure 1. The running time of our plowing algorithm lies between $O(n \log n)$ and $O(n^3 \log n)$ depending on how much it changes the sketch. Consecutive plowing operations may use different directions. Our plowing algorithm is more general than the one presented in [20] because it works on the symbolic instead of the geometric level and hence can change wire geometries more cleverly, secondly, the plow can be a line segment of arbitrary orientation and is not constrained to be iso-oriented, and thirdly, the plow can move in an arbitrary direction and not just parallel to the coordinate axes.

This paper is structured as folows. In section I, we give the relevant definitions and state the problem and the results precisely. In section II, we prove the correctness of our algorithm and describe an efficient implementation of it. The main novel idea in the implementation is an efficient data structure for the maintenance of the capacities and densities of all cuts. In section III we describe several extensions (in particular, plowing) and in section IV we offer a short conclusion.

I. Definitions and Results

A *sketch* is a triple (F, W, P) consisting of a finite set F of *features*, which are points (= point feature) and open straight line segments (= line feature), a finite set W of *wires*, which are simple paths in the plane, and a partition P of the features F. Each block of the partition is called a *module*. Figure 2 shows an example of a sketch. When the partition P is understood we will refer to a pair (F, W) as a sketch. The features and wires of a sketch must satisfy the following conditions:

(1) Distinct features do not intersect and the endpoints of each line feature are point features.
(2) No wire may cross itself.
(3) Each wire touches exactly two features, which are point features lying at the endpoints of the wire. They are called the *terminals* of the wire.
(4) There are two line features, called the left and right bar, which are infinite vertical lines and lie to the left and right of all other features.

A *point in a sketch* is a point lying on a feature. Modules form the rigid part of a layout and wires represent the flexible interconnections.

Sketches comprise the information of placement and global routing. A (detailed) *routing* of a sketch (F, W, P), $W = \{p_1, \ldots, p_m\}$, is a sketch (F, W', P), $W' = \{q_1, \ldots, q_m\}$,

such that q_i is *homotopic* to p_i, i.e., p_i and q_i have the same endpoints and p_i can be transformed continuously into q_i without moving its endpoints and without allowing its interior to touch a feature in F, and such that the q_i's satisfy the constraints of the particular wiring model used. We consider two wiring models in this section, the grid model and the free model, and comment on the knock-knee model in the conclusion.

In the *grid model* wires are vertex-disjoint paths in the rectangular grid of unit-spacing. Figure 3 (4) shows a routing of the (compacted) sketch of Figure 2 in the grid model. In the *free model* wires are arbitrary paths in the plane satisfying the following minimum separation constraints (for a point set $T, U(T) = \{x;\ dist(x,t) < 1/2$ for some $t \in T\}$ denotes the open 1/2-neighborhood of T):

1) $U(q_i) \cap U(q_j) = \emptyset$ for $i \neq j$
2) $U(q_i) \cap U(f) = \emptyset$ where f is any feature which is not a terminal of q_i
3) $U(q_i)$ is simply connected.

Figure 5 shows a routing of the compacted sketch of Figure 2 in the free model.

A *cut* is any open line segment connecting two points of the sketch, say $p = (x_p, y_p)$ and $q = (x_q, y_q)$, and not intersecting any feature. We denote the cut with endpoints p and q by $\overline{pq}$. The *density* of cut $\overline{pq}$ is the number of crossings of $\overline{pq}$ by wires which are enforced by the topology of the sketch, cf. Figure 6. Crossings of $\overline{pq}$ which can be removed by deforming the wires do not contribute to the density. The *capacity* of a cut in the free model is the Euclidean length (of the corresponding closed segment) minus one and the capacity in the grid model is given by $\max\{|x_p - x_q|, |y_p - y_q|, 1\} - 1$. A cut is called *safe* if its density does not exceed its capacity and it is called *tight* or *saturated* if its density is equal to its capacity. The following theorem was proved by Cole/Siegel and Leiserson/Maley for the grid model and by Gao et al. and Maley for the free model.

Theorem 1. *A sketch has a routing iff all cuts of the sketch are safe.*

Actually, the results are slightly stronger. Let us call a cut $\overline{pq}$ *critical*, if either p and q are point features or at most one of them lies on a line feature and the line segment $\overline{pq}$ is perpendicular to that line feature. Then a sketch is routable iff all critical cuts are safe.

We are now ready to define the (one-dimensional) compaction problem. The goal of compaction is to displace the modules in x-direction such that the resulting sketch is routable and has minimal x-width (the x-width of a sketch is the horizontal distance between the left and right bar). Let $S = (F, W, P)$ be a *routable sketch*. We denote the displacement of feature $f \in F$ by $d(f)$ and call the vector $d \in \mathbb{R}^F$ of displacements a *configuration*. Of course, not all displacements make sense. Firstly, features in the same module must be displaced by the same amount and therefore we must have $d(f) = d(g)$ for any two features in the same module. Secondly, features should not cross over during compaction and we therefore must have $x_p + d(f) < x_q + d(g)$ for any two points $p = (x_p, y_p)$ and $q = (x_q, y_q)$ where $x_p < x_q$ and $y_p = y_q$ and p lies on a feature f and q lies on a feature g. Let d be a configuration satisfying the two constraints above. We can now define the sketch $S(d)$ in a natural way. A point p on feature f with coordinates (x_p, y_p) in the initial sketch has coordinates $(x_p + d(f), y_p)$ in $S(d)$ and the wires in $S(d)$ have the

"same" homotopies as in S; cf. [13] for a more precise definition. Figures 4 and 5 show compacted versions of the sketch of Figure 2. The *configuration space* $C(S) \subseteq \mathbb{R}^F$ of a sketch S consists of all configurations d such that d satisfies the two constraints above and $S(d)$ is routable. Note that the zero-vector 0 belongs to $C(S)$ since the sketch S is assumed to be routable. The *essential configuration* space $C_0(S)$ of a sketch S consists of that connected component of $C(S)$ which contains the zero-vector, i.e., a configuration d belongs to $C_0(S)$ if the sketch $S(d)$ can be obtained by continuously deforming the sketch S going only through routable intermediate sketches. The compaction problem can now be formalized as follows.

(One-Dimensional) *Compaction Problem* [13]
Input: A routable sketch $S = (F, W, P)$
Output: A configuration $d \in C_0(S)$ such that $S(d)$ has minimal x-width

Theorem 2 [13] .
Let $S = (F, W, P)$ be a routable sketch. Then the essential configuration space $C_0(S)$ of the sketch S is a convex polyhedron.

Maley gave an explicit description of the polyhedron $C_0(S)$ and made it the basis of his solution for the compaction problem. In order to state his result we need some more notation. Let us assume that the wires in a sketch are given by polygonal paths. For a wire $w \in W$, let b_w be the number of line segments in the polygonal path for w, let $b = \sum_{w \in W} b_w$, and let $m = |F|$ be the number of features.

Theorem 3 [13] .
The compaction problem can be solved in time $O(m^4 + m^2 b \log mb) = O(n^4)$ where $n = m + b$ is the size of the input sketch.

In this paper we improve upon theorem 3 and show

Theorem 4. *The compaction problem can be solved in time $O(mb + (m^2 + k) \log m) = O(n^3 \log n)$, where $n = m + b$, and k is the number of times that a module moves across a cut during the compaction process; $k = O(m^3)$ always.*

We prove theorem 4 in section II. The quantity k measures in a certain sense how much the compaction algorithm changes the sketch. k can be as large as $O(m^3)$, and as small as 0 (if the input sketch has already minimal x-width). In practical compaction problems, where the input sketch is frequently nearly optimal, we expect k to be much smaller than $O(m^3)$. Under the assumption that a cut is crossed by only $O(1)$ features on average we have $k = O(m^2)$ and hence a running time of $O(n^2 \log n)$.

II. The new compaction algorithm

Let $S = S(0)$ be a routable sketch. Recall that there are two vertical bars enclosing the sketch. The idea underlying our algorithm is very simple. We move the right bar to the

left. Whenever a tight cut between an already moving feature and a still motionless feature arises the motionless feature starts to move along with the right bar. We continue in this fashion until the left bar starts to move. At this point we have minimized the x-width of the sketch. We will now describe the algorithm in more detail.

At each position P ($= x$-coordinate) of the right bar we have a partition of the features into two sets L and R. Initially, R consists only of the right bar. In general, R consists of a certain set of modules, i.e., if a feature f belongs to R then the entire module containing f is a subset of R. Moreover, there exists a rooted tree on the modules in R such that the right bar is the root of the tree and such that for every non-root M there is a point p on M and a point q on the parent of M such that $\overline{pq}$ is a tight cut in the current configuration and p has smaller x-coordinate than q. We will frequently use "p lies to the left of q" instead of "p has smaller x-coordinate than q".

The *current configuration* is defined implicitly by the algorithm below. The initial configuration is the initial sketch S. Suppose now that the current configuration $d(P)$ is defined for some position P of the right bar and let $\Delta > 0$ be such that the partition (L, R) does not change in the interval $(P - \Delta, P)$. Then the configuration $d(P - \Delta)$ is obtained from $d(P)$ by moving all features in R to the left by the amount Δ. For ease of notation we write S_P instead of $S(d(P))$. It remains to describe when and how the partition (L, R) changes.

Lemma 1. *Let P be the current position of the right bar, and let S_P be routable and such that there is no tight critical cut $\overline{pq}$ with $p \in f \in L, q \in g \in R$ and p to the left of q in S_P. Let $\Delta > 0$ be minimal such that a non-vertical critical cut becomes tight in $S_{P-\Delta}$.*

a) $S_{P-\delta}$ is routable for $0 \leq \delta \leq \Delta$.

b) Let $\overline{pq}$ be a cut which becomes tight in $S_{P-\Delta}$. Then $p \in f \in L, q \in g \in R$ and p lies to the left of q in $S_{P-\Delta}$.

Proof: a) Assume otherwise. Let $\Delta_1 = \inf\{\delta;\ S_{P-\delta}$ is not routable$\}$. Then $S_{P-\Delta_1}$ is routable and hence $\Delta_1 < \Delta$. Let $\overline{pq}$ be a cut which is oversaturated in $S_{P-\Delta_1-\delta}$ for all sufficiently small δ. We may assume w.l.o.g. that $\overline{pq}$ is critical. Assume first that $\overline{pq}$ is a cut in $S_{P-\Delta_1}$. Then $\overline{pq}$ is safe in $S_{P-\Delta_1}$. Also, if p is not to the left of q then the capacity of $\overline{pq}$ increases when going to $S_{P-\Delta_1-\delta}$ and hence $\overline{pq}$ cannot become oversaturated. Thus p must be to the left of q in $S_{P-\Delta_1}$ and therefore the cut $\overline{pq}$ became tight in $S_{P-\Delta_1}$, a contradiction to the choice of Δ_1. Assume next that $\overline{pq}$ is not a cut in $S_{P-\Delta_1}$. Then there are features $q_1, q_2, \ldots, q_n$ lying on the line segment $\overline{pq}$ in $S_{P-\Delta_1}$, cf. Figure 8. The cuts $\overline{pq_1}, \overline{q_1q_2}, \ldots, \overline{q_nq}$ are not oversaturated in $S_{P-\Delta_1}$ and hence the cut $\overline{pq}$ is not oversaturated in $S_{P-\Delta_1-\delta}$ by the proof of the claim in lemma 7 of [Maley 85]. So, we derived a contradiction.

b) Assume otherwise. Then either $f, g \in L, f, g \in R$ or $f \in R$ and $g \in L$. Also, $\overline{pq}$ is a cut in $S_{P-\Delta}$ and hence in $S_{P-\Delta+\delta}$ for all sufficiently small $\delta > 0$. In all three cases the density of the cut $\overline{pq}$ is the same in $S_{P-\Delta+\delta}$ and in $S_{P-\Delta}$, in the first two cases the capacity is also the same and in the third case it is even larger than in $S_{P-\Delta}$. Thus the cut $\overline{pq}$ cannot become tight in $S_{P-\Delta}$, a contradiction. ∎

Let P and Δ be as in the premise of lemma 1. Let R' be defined by program 1.

---

(1) $\mathbb{R}' \leftarrow R; L' \leftarrow L;$
(2) **while** $\exists$ tight critical cut $\overline{pq}$ with $p \in f \in L', q \in g \in R'$
and p to the left of q in $S_{P-\Delta}$
(3) **do** $R' \leftarrow \{f'; f$ and f' belong to the same module$\}$
(4) $\quad L' \leftarrow F - R'$
(5) **od**

Prog. 1

Then $S_{P-\Delta}$ is routable by part a) of lemma 1 and there is no tight critical cut $\overline{pq}$ with $p \in f \in L', q \in g \in R'$ and p to the left of q in $S_{P-\Delta}$, i.e., the premise of lemma 1 is again satisfied.

It remains to define a rooted tree on the modules in R' with the desired properties. Let $\overline{pq}$ be a tight critical cut with $p \in f \in L', q \in g \in R'$ and p to the left of q in $S_{P-\Delta}$. We make the module containing g the parent of the module containing f. Also, if the cut $\overline{pq}$ became tight in $S_{P-\Delta}$ then there might be a point $r \in h \in R$ such that p lies on $\overline{rq}$ in $S_{P-\Delta}$, $\overline{rq}$ is a tight cut and the module containing g is the parent of the module containing h. In this case, the cut $\overline{rp}$ is also tight and we make the module containing f the new parent of the module containing h; cf. Figure 8.

Theorem 5. *The algorithm above solves the compaction problem.*

Proof: Let c_0 be the final configuration computed by the algorithm. Then there is a sequence $M_1, \ldots, M_k$ of modules such that M_1 is the right bar, M_k is the left bar, and for every $i \geq 2$ there is a point p_i in M_i and a point q_i in M_{i-1} such that p_i lies to the left of q_i and the cut $\overline{p_i q_i}$ is tight in $S(c_0)$. The minimality of c_0 follows now from

Lemma 2. *Let $d \in C_0$ be an arbitrary configuration. Let $d_i = x_{q_i} + d(q_i) - (x_{p_i} + d(p_i))$ and $c_i = x_{q_i} + c_0(q_i) - (x_{p_i} + c_0(p_i))$ be the horizontal distance of the points p_i and q_i in the configurations d and c_0 respectively. Then $d_i \geq c_i$ for all i and the x-width of d is at least the x-width of c_0.*

Proof: Assume otherwise, say $d_i < c_i$ for some i. For $0 \leq \lambda \leq 1$, let $d(\lambda) = (1-\lambda)c_0 + \lambda d$. Then $d(\lambda) \in C_0$ since C_0 is convex. Also the difference between the x-coordinates of q_i and p_i in $S(d(\lambda))$ is $(1-\lambda)c_i + \lambda d_i < c_i$ for $\lambda > 0$. Next note that the line segment $\overline{p_i q_i}$ is a cut in $S(c_0)$ and hence in $S(d(\lambda))$ for $\lambda > 0$ and sufficiently small. The density of $\overline{p_i q_i}$ in $S(c_0)$ and $S(d(\lambda))$ is the same and the cut is saturated in $S(c_0)$. So the cut is oversaturated in $S(d(\lambda))$ for $\lambda > 0$ and small. This is a contradiction to $d(\lambda) \in C_0$. ∎

The correctness of our algorithm is now established. We turn to its implementation next. First we introduce the concepts of cut changing (C) and tightening (T) events and show how to compute them efficiently. Then we reformulate the compaction algorithm in terms of C-events and T-events.

Let f and g be features with one of them, say f, being a point feature. If g is also a point feature then $\overline{fg}$ denotes the line segment connecting f and g, if g is a line feature then $\overline{fg}$ denotes the line segment through f perpendicular to g. A line segment $\overline{fg}$ is a cut if it intersects no other feature.

A C-event occurs when a cut appears or disappears. Note that the next C-event is the next position of the right bar where two existing cuts $\overline{fg}$ and $\overline{gh}$ become collinear (see Figure 9).

The *T-event* of a cut $\overline{fg}$ is the next position of the right bar where $\overline{fg}$ becomes tight. This position is computed under the assumption that the density of $\overline{fg}$ and the partition (L, R) does not change until the event occurs. Note that the T-event only exists for cuts connecting a feature in L with a feature in R. For cuts connecting two features in the same set the capacity does not change and hence no T-event exists (or equivalently, takes place at $-\infty$).

We keep all C-events and T-events in a priority queue, called the *global event queue* Q. The next event is the minimal element of Q. Let P and Δ satisfy the premise of Lemma 1. Then it is clear that the next event occurs no later than at position $P - \Delta$, because if no C-event occurs before $P - \Delta$ then a T-event will occur at $P - \Delta$.

Knowing the density of a cut $\overline{fg}$ and the current status of its two endpoints (element of L or R) the T-event of $\overline{fg}$ can easily be computed in time $O(1)$. To compute the C-events we have to work harder.

We maintain for every point feature f a special data structure which we call the *C-structure* of f. It is defined as follows. Consider all cuts incident to f. If we extend these cuts to straight lines, they partition the plane into sectors (or wedges). (cf. Figure 10)

These sectors will change their size during the compaction, since moving point features in R will cause the corresponding lines to move too. Let $S(f, g, h)$ be the sector between the lines through g and h (cf. Figure 10). Every time a sector collapses, i.e., points f, g and h become collinear, a C-event occurs.

Now compute for every sector $S(f, g, h)$ the position of the right bar $P(f, g, h)$ when this sector collapses. We call $P(f, g, h)$ the *collapsing time* of $S(f, g, h)$. The collapsing time is computed under the assumption that the status (= element of L or R) of the three points f, g, h will not change before the event.

$P(f, g, h)$ is either $-\infty$ or the solution of a simple system of 3 linear equations which can be solved in time $O(1)$. For example, in the situation of figure 10 with $f \in L, h \in R$ and $g \in L$ we have $P(f, g, h) = \delta$ with

$$\begin{vmatrix} 1 & 1 & 1 \\ x_f & x_g & x_h + \delta \\ y_f & y_g & y_h \end{vmatrix} = 0$$

where x_q, y_q are the initial coordinates of point feature $q \in \{f, g, h\}$. All other situations lead to similar equations. We store each sector $S(f, g, h)$ according to its collapsing time $P(f, g, h)$ in a local priority queue at f. Then we have

Lemma 3. *a) The C-structure of a point feature f needs space $O(m)$ and can be constructed in time $O(m \log m)$.*

b) The minimal event of all local priority queues is the next C-event.

Every time a C-event occurs, i.e., a sector $S(f,g,h)$ collapses, the C-structure of f has to be updated as follows. Let $S(f,x,h)$ be the sector left of $S(f,g,h)$ and let $S(f,g,y)$ be the sector right of $S(f,g,h)$ (cf. Figure 10).

- remove $S(f,x,h)$, $S(f,g,h)$ and $S(f,g,y)$ from the local priority queue
- compute $P(f,x,h)$ and $P(f,h,y)$
- insert $S(f,x,h)$ and $S(f,h,y)$ into the local queue according to their collapsing time

If a cut $\overline{fg}$ disappears it has to be deleted from the C-structures of f and g (if point features). Furthermore the T-event of $\overline{fg}$ must be removed from the global priority queue Q.

If a cut $\overline{fg}$ appears it has to be inserted into the C-structure of f and g and the density and the T-event of $\overline{fg}$ must be computed.

All of this can be done in time $O(\log n)$ (We will show later how the density can be computed in time $O(\log n)$).

If a T-event for some potential cut $\overline{fg}$ occurs f is removed from L and put into R. This may change the collapsing times of all sectors in the C-structure of f. Therefore we rebuild the entire C-structure of f in this case which has cost $O(m \log m)$.

Furthermore we have to update the collapsing time of the two sectors incident to f for all C-structures of point features $h \neq f$. This again has cost $O(m \log m)$ since we perform $O(m)$ update operations on local priority queues.

Every time a new cut $\overline{fg}$ appears its density has to be computed. Let $\overline{fh}$ and $\overline{hg}$ be the two cuts that became collinear (cf. figure 11). Then we have

$$density(\overline{fg}) = density(\overline{fh}) + density(\overline{hg}) + \#\text{nets with terminal } h - 2 \cdot \#\text{turning nets}$$

To compute the number of turning nets we maintain the sketch during the compaction as follows. Consider the nets in the sketch as tight rubber bands. Then each net forms a polygonal path whose edges are cuts and boundary edges of modules and whose vertices are corners of modules. Now assign to every cut or edge $\overline{fg}$ the bundel of nets that use the line segment $\overline{fg}$ in the rubber band model.

For every corner h of a module we store all cuts and edges incident to h in a list $L(h)$ in clockwise ordering and associate with each entry $\overline{fh}$ the size of its bundel $b(\overline{fh})$. When two cuts $\overline{fh}$ and $\overline{hg}$ become collinear and create the new cut $\overline{fg}$ the number of turning nets can be computed as follows (cf. figure 12).

Let a be the number of nets using cuts before $\overline{fh}$ in the list $L(h)$ and b be the number of nets using cuts after $\overline{hg}$ in $L(h)$, i.e.

$a = \sum\{b(\overline{xh}) \mid \overline{xh} \in L(h) \text{ and } \overline{xh} < \overline{fh}\}$
and $b = \sum\{b(\overline{hy}) \mid \overline{hy} \in L(h) \text{ and } \overline{hy} > \overline{hg}\}$.

Then the number of turning nets is $\min(a, b)$, since wires do not cross. Note that a and b can be computed in time $O(\log n)$ if the lists $L(h)$ are implemented as balanced search trees. If a C-event occurs the above defined data structure for maintaining the sketch must be updated. If a cut $\overline{fg}$ disappears the corner h of a module comes to lie on it (cf. figure 13) and the following actions have to be performed:

$$b(\overline{fh}) \leftarrow b(\overline{fh}) + b(\overline{fg})$$

$$b(\overline{hg}) \leftarrow b(\overline{hg}) + b(\overline{fg})$$

delete $\overline{fg}$ from $L(f)$

delete $\overline{fg}$ from $L(g)$

All of this takes time $O(\log n)$.

The case where a new cut $\overline{fg}$ is created is a bit more complicated. Assume that cuts $\overline{fh}$ and $\overline{hg}$ become collinear. Before this event happens the situation is as shown in figure 14.

In this figure, a_1 is the number of nets using cuts before $\overline{fh}$, $a_2 = b(\overline{fh})$ and a_3 is the number of nets using cuts between $\overline{fh}$ and the horizontal line through h.
b_1 is the number of nets using cuts after $\overline{hg}$, $b_2 = b(\overline{hg})$ and b_3 is the number of nets using cuts between the horizontal line through h and $\overline{hg}$.

All the numbers can be computed in time $O(\log n)$ if $L(h)$ is implemented as a balanced tree. For example $a_1 = \sum\{b(\overline{xh}) \mid \overline{xh} \in L(h) \text{ and } \overline{xh} > \overline{fh}\}$.

Our goal is to compute $b(\overline{fg})$ and the new values for $b(\overline{fh})$ and $b(\overline{hg})$ (we denote these values by $b'(\overline{fg}), b'(\overline{fh}), b'(\overline{hg})$). It is easy to see that
$b'(\overline{fh}) = b(\overline{fh}) - b'(\overline{fg})$ and $b'(\overline{hg}) = b(\overline{hg}) - b'(\overline{fg})$.

The bundel of the new cut $\overline{fg}$ is the intersection of the bundels of $\overline{fh}$ and $\overline{hg}$. $b'(\overline{fg})$ is the cardinality of this intersection. The following case analysis for the computation of $b'(\overline{fg})$ relies heavily on the assumption the nets do not cross. A generalization to knock-knee mode is discussed at the end of this paper.

case 1: $a_2 = 0$ or $b_2 = 0$
i.e., at least one of the bundels of $\overline{fh}$ or $\overline{hg}$ is empt. Then $b'(\overline{fg}) = 0$.

case 2: $a_2 \neq 0$ and $b_2 \neq 0$
Assume w.l.o.g. that $a_3 \geq b_3$
(the other case is symmetric, just exchange the a's and b's)

2.1: $a_3 > b_2 + b_3$
then there is no net using both $\overline{fh}$ and $\overline{hg}$ and hence $b'(\overline{fg}) = 0$

2.2: $a_3 \leq b_2 + b_3$
2.2.1: $a_1 \geq b_1$

Then all nets in the bundel of $\overline{fh}$ are also contained in the bundel of $\overline{hg}$ and we have $b'(\overline{fg}) = a_2$
2.2.2: $a_1 < b_1$
In this case there are $b_1 - a_1$ nets in the bundel of $\overline{fh}$ that are not contained in the bundel of $\overline{hg}$ and we have $b'(\overline{fg}) = a_2 - b_1 + a_1$.

This completes the case analysis. We have shown that the additional cost for maintaining the sketch is $O(\log n)$ per C-event. Thus the total cost of a C-event is $O(\log n)$.

Note that both data structures presented above (for computing C-events and for maintaining the sketch) have space requirement linear in the number of all currently existing cuts which is $O(n^2)$ in the worst case but which can be exepcted to be much smaller in practice.

We are now ready to formulate the entire compaction algorithm.

Preprocessing: (* takes time $O(bn + n^2 \log n)$*)

(1) compute the visibility graph $G = (V, E)$ of the input sketch
(2) **for** all cuts $\overline{fg} \in E$
(3) **do** compute the density
(4) the capacity
(5) **od**
(6) **for** all cuts and edges $\overline{fg}$
(7) **do** compute the size of its bundel $b(\overline{fg})$
(8) insert $\overline{fg}$ into $L(f)$
(9) insert $\overline{fg}$ into $L(g)$
(10) **od**
(11) **for** all point features f
(12) **do** construct the C-strructure of f **od**

Initialization: (* takes time $O(n \log n)$*)

(1) $R \leftarrow$ {"right bar"}
(2) $L \leftarrow$ all features not in R
(3) initialize the global event queue with all C-events and T-events $\neq -\infty$
(* Note that then are only $O(m)$ events in the beginning *)

Compaction: (* takes time $O((n^2 + k) \log m)$*)

consider the minimal event in the global event queue Q; it is either a C- or a T-event.

C-event: point features f, g, h become collinear

-- update the C-structure of f, g, h as described before
(* takes time $O(\log n)$*)
– update the sketch as described before
(* takes time $O(\log n)$*)
case 1: the cut $\overline{fg}$ disappears
– delete the T-event of $\overline{fg}$ from Q
case 2: a new cut $\overline{fg}$ is created
– compute its density and T-event

and insert it into Q
(* both cases need time $O(\log n)$*)

T-event: let $\overline{fg}, f \in L, g \in R$ be the cut that became tight.
- $R \leftarrow R \cup \{f\}$; $L \leftarrow L \backslash \{f\}$
- update the C-structure of f and of all point features visible from f
(* takes time $O(n \log n)$*)
- update the global event queue Q (delete old C-events,
insert new C-events) (* takes time $O(n \log n)$*)
- compute the T-events for all cuts incident to f, insert them into Q

(* the total cost of a T-event is $O(n \log n)$*)

Theorem 6. *The above algorithm solves the compaction problem in time $O(bn + (n^2 + k) \log n)$ and space $O(V)$, where k is the number of C-events and V is the maximal size of the visibility graph of the sketch.*

Proof: The space bound follows from the fact that both the data structure for computing the events and that for maintaining the sketch need space proportional to the size of the visibility graph of the current sketch. We have already seen that the algorithm spends time $O(\log n)$ in the C-event case and $O(n \log n)$ time in the T-event case. Now the time bound follows from the observation that there are at most $O(n)$ T-events (every time a T-event occurs a feature moves from L to R). ∎

The number of C-events may be as large as $O(m^3)$. However, if the initial sketch is already optimal then $k = 0$ and if the compaction process brings about only local changes of the layout then $k = O(m^2)$ since cuts are crossed by only $O(1)$ features in this case. On the other hand, a large value of k can only occur if the compaction changes the layout globally. In this case, the increased running time is well spent.

III. Extensions

In this section we present extensions and modifications of our compaction algorithm which can be implemented without great effort.

1. Maximum distance constraints

In practice there are often constraints which postulate that the distace of two layout components must not exceed a certain value. They are either user-defined or result from the technology. With a small modification our algorithm can also handle such maximum distance constraints. We only have to define the T-event of a maximum distance constraint, say between features f and g, as the next position of the right bar where the distance reaches its maximal value. If the T-event of a maximal distance constraint between f and g occurs then $f \in R, g \in L$ and f is to the left of g. We add g to R at this point. All other parts of the algorithm remain unchanged.

2. Compaction in arbitrary directions

None of the data structure presented uses the fact that compaction is done in x-direction. So we may allow arbitrary directions; we only have to change the part of the algorithm where the collapsing times of sectors of C-structures are computed, because now not only the x-coordinates of point features, but also the y-coordinates change over time. But the collapsing time of a sector is still the solution of a simpe system of linear equations and can be computed in time $O(1)$.

3. Plowing

Plowing is a powerful concept of the Magic Layout Editor [17]. It can be used interactively to rearrange the geometry of a cell, compact a sparse layout or to create new space in a dense layout. The user places a plow (horizontal or vertical line segment) into the layout and gives the direction and distance the plow is to move. According to these specifications the plow is then moved through the layout and the material behind it is stretched (if necessary) and the material in front of it is compressed. For details see [20].

Our compaction method can support plowing as well. After the user has defined the plow and the direction and distance it is to move, the set R is initialized with the plow and the global event queue is initialized with all C-events and T-events $\neq -\infty$. Since there are at most $O(n)$ such events (local visibility), the initialization step takes time $O(n \log n)$.

Now the compaction algorithm is executed as described before. Its running time is bounded by $O(k \log n)$ where k is the number of C-events occuring during the plowing operation.

The data structures for computing the events and for maintaining the sketch have to be computed only once for an entire sequence of plowing operations since they do not depend on the compaction direction and the initial values of L and R (as discussed in part 2 of this section); thus our plowing procedure can be used as an efficient interactive tool. The cost of an entire sequence of consecutive plowing operations only depends on how much the sketch is changed by this sequence.

The plowing operation supported by our compaction algorithm is more general than the one presented in [20]. Firstly, it works on the symbolic level instead of the geometric level and hence can change wire geometries more cleverly, secondly, the plow can be a line segment of arbitary orientation and is not constrained to be iso-oriented, and thirdly, the plow can move in an arbitrary direction and not just parallel to the coordinate axes.

4. Compaction of channels

If we use our algorithm for minimizing the length of a channel of given width its performance is better than in the general case. Assume that a horizontal channel configuration is to be compacted in x-direction, then there are no C-events since no cut disappears and none is created. Thus we have $k = 0$ in theorem 6 and the running time is $O(n^2 \log n)$.

Of course, the algorithm of Leiserson/Pinter solves this problem in linear time.

IV. Conclusions

We presented a compaction algorithm which achieves optimal one-dimensional compaction with jog-insertion. Its running time is $O(n^2 \log n)$ in the best case and deteriorates to $O(n^3 \log n)$ if the compaction process changes the layout dramatically. The algorithm is conceptually very simple; only its implementation is nontrivial. The algorithm follows the push (minimum distance constraints) and pull (maximum distance constraints) paradigm. Another recent application of this paradigm can be found in [15].

We discussed the compaction problem for one-layer routing. The algorithm can be applied to multi-layer routings by applying it to each layout plane separately. Of course, the vias and the layer assignment must be part of the input sketch in this case.

It is very likely that our algorithm also applies to routing in knock-knee mode (wires are edge-disjoint paths on a grid). Kaufmann/Mehlhorn have shown that a sketch can be routed if all cuts are safe and the free capacity (= capacity - density) of every cut is even. Because of the evenness condition the routability condition is only sufficient but not necessary in this case. Suppose now that we are given a sketch which satisfies the routability condition. Then the free capacity of every nonsaturated cut is a multiple of two and hence one should be able to move the right bar by a multiple of two. If the resulting sketch were still even, a fact which we have not shown yet, then we could iterate and our algorithm would find the sketch of smallest x-width also satisfying the routability condition (of course, this may not be the routable sketch of smallest x-width). The algorithms of Kaufmann/Mehlhorn and Brady/Brown can then turn the sketch into a 4-layer detailed routing. In this way, layer assignment would be postponed until after compaction.

V. References

[1] Brady, M., and Brown, P., "VLSI Routing: Four Layers Suffice", Advances in Computing Research, Ed. F. Preparata, 1984, pp. 245–258

[2] Cole, R., and Siegel, A., "River routing every which way, but loose", Proceedings of the 25th Annual Symposium on Foundations of Computer Science, October 1984, pp. 65–73

[3] Dunlop, A.E., "SLIP: symbolic layout of integrated circuits with compaction", Computer Aided Design, Vol. 10, No. 6, 1978, pp. 387–391

[4] Hsueh, M.Y., "Symbolic Layout and Compaction of Integrated Circuits", Ph.D. thesis, EECS Division, University of California, Berkeley, 1979

[5] Gao, S., Jerrum, M., Kaufmann, M., Mehlhorn, K., Rülling, W., Storb, C., "On homotopic river routing", BFC 87, Bonn, 1987

[6] Kaufmann, M. and Mehlhorn, K., "Local Routing of Two-Terminal Nets", 4th STACS 87, LNCS 247, pp. 40–52

[7] Kedem, G., and Watanabe, H., "Optimization techniques for IC layout and compaction", Techn. Report 117, Computer Science Department, University of Rochester, 1982

[8] Leiserson, C.E., and Maley, F.M., "Algorithms for routing and testing routability of planar VLSI layouts", Proceedings of the 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 69–78

[9] Leiserson, C.E., and Pinter, R.Y., "Optimal placement for river routing", SIAM Journal on Computing, Vol. 12, No. 3, 1983, pp. 447–462

[10] Lengauer, T., "Efficient algorithms for the constraint generation for integrated circuit layout compaction", Proceedings of the 9th Workshop on Graphtheoretic Concepts in Computer Science, 1983

[11] Lengauer, T., and Mehlhorn, K., "The HILL system: a design environment for the hierarchical specification, compaction, and simulation for integrated circuit layouts", Proceedings, Conference on Advanced Research in VLSI, 1984

[12] Lengauer, T., "On the solution of inequality systems relevant to IC layout", Journal of Algorithms, Vol. 5, No. 3, 1984, pp. 408–421

[13] Maley, F.M., "Compaction with Automatic Jog Insertion", 1985 Chapel Hill Conference on VLSI

[14] Maley, F.M., personal communication, 1987

[15] Mehlhorn, K.,and Rülling, W., "Compaction on the Torus", Techn. Report, FB10, Informatik, Universität des Saarlandes, Saarbrücken, Dec. 87

[16] Mehlhorn, K., "Data Structures and Algorithms", Vol. 3, Springer Publ. Company, 1984

[17] Ousterhout, J.K., Hanachi, G., Mayo, R.N., Scot, W.S., Taylor, G.S., "The Magic VLSI Layout System", Proceedings of the 21st Design Automation Conference, June 1984

[18] Pinter, R.Y., "The Impact of Layer Assignment Methods on Layout Algorithms for Integrated Circuits", Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, 1982

[19] Rülling, W., "Einführung in die Chip-Entwurfssprache HILL", Technischer Bericht 04/1987, SFB124, Universität des Saarlandes, 1987

[20] Scott, W.S., and Ousterhout, J.K., "Plowing: interactive stretching and compaction in Magic", Proceedings of the 21th Design Automation Conference, 1984, pp. 166–172

[21] Williams, J.D., "STICKS — a graphical compiler for high level LSI design", National Computer Conference, 1978, pp. 289–295

[22] Xiong, X-M. "Optimized One-Dimensional Compaction of Building-Block Layout", Berkeley Memorandum UCB/ERL M87/45, 1987

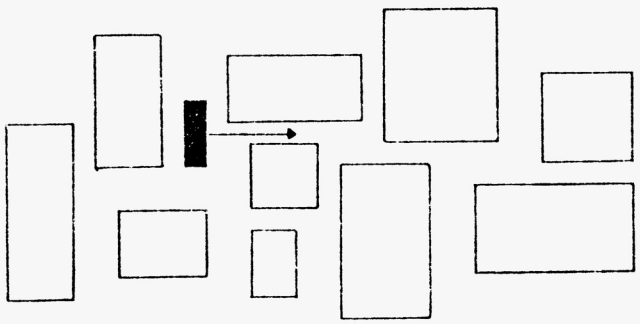

Figure 1: Example of a plowing operation.

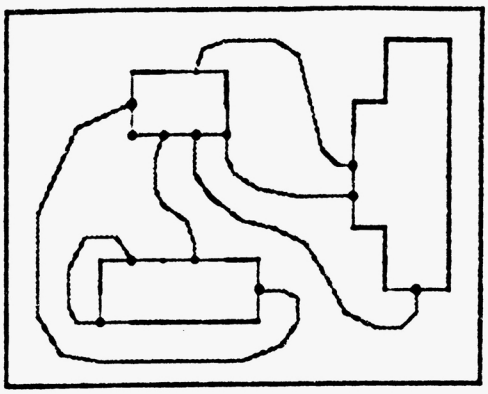

Figure 2: A typical sketch.

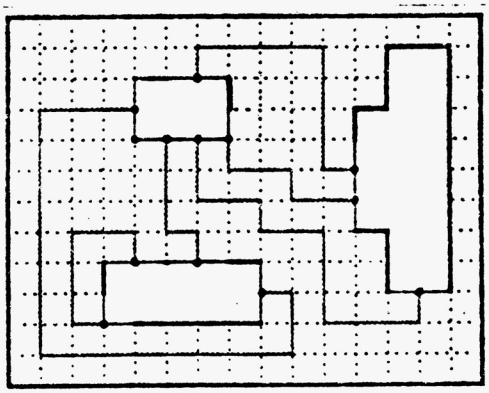

Figure 3: A routing of the sketch in figure 2 using wires of minimal length. The width is 15.

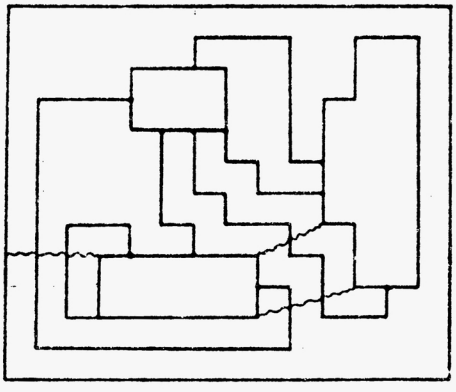

Figure 4: A routed and compacted version of the sketch of figure 2 (grid model); tight cuts are shown as wiggled lines. The width is 14.

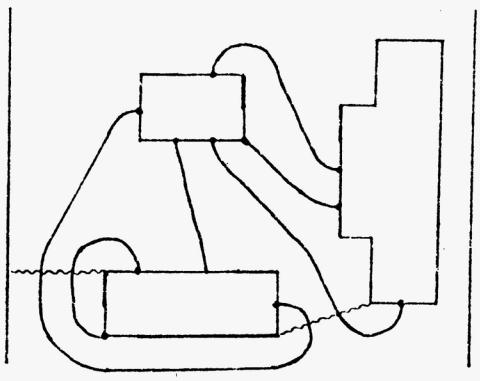

Figure 5: A routed and compacted version of the sketch of figure 2 (free model). The width is $11 + 2\sqrt{2}$.

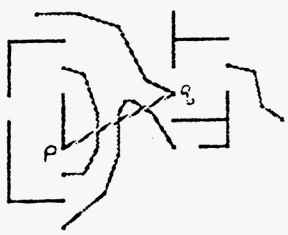

Figure 6: A portion of a sketch with cut $\overline{pq}$. The flow across $\overline{pq}$ is 1.

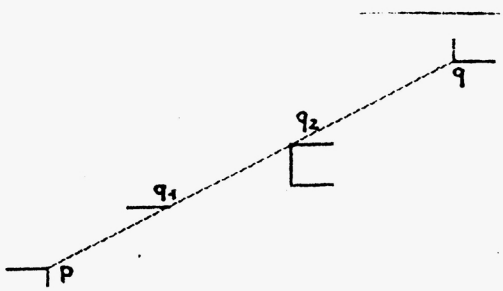

Figure 7: The features q_1, q_2, q_3, ... on the line segment $\overline{pq}$.

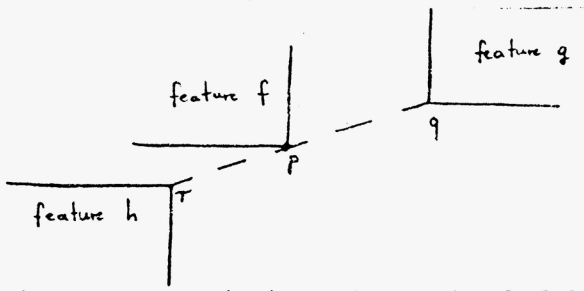

Figure 8: $r(p,q)$ lies on feature $h(f,g)$. The features h and g belong to R and the cut $\overline{rq}$ is tight. When $p \in f \in L$ comes to lie on $\overline{rq}$ in $S_{P-\Delta}$ both cuts $\overline{rp}$ and $\overline{pq}$ are tight.

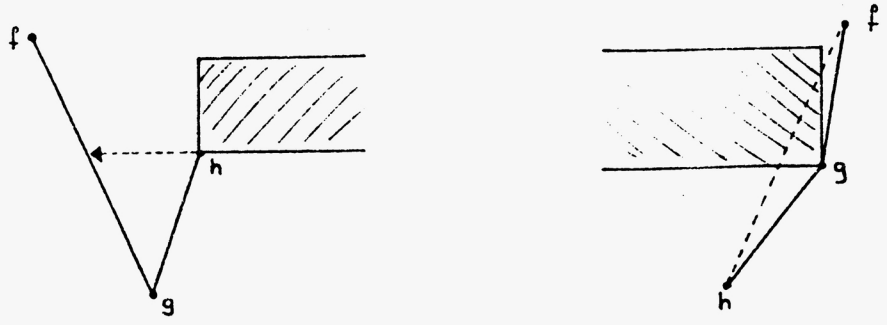

Figure 9

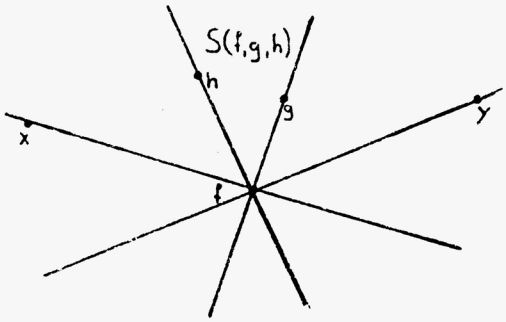

Figure 10

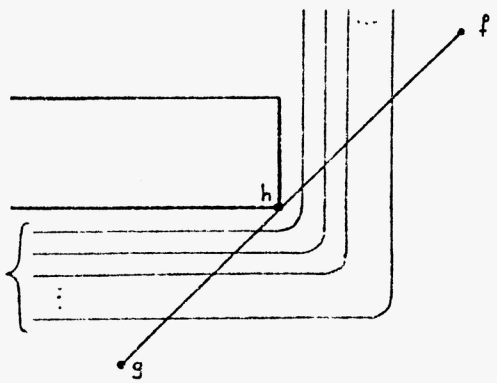

Figure 11

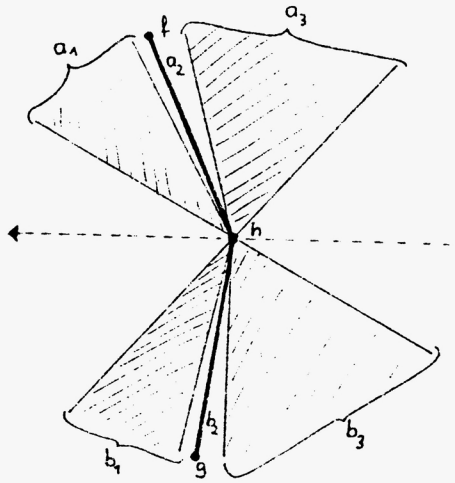

Figure 12

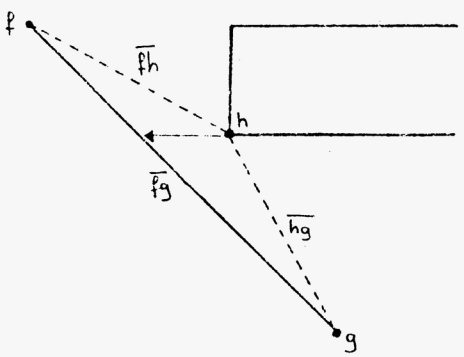

Figure 13

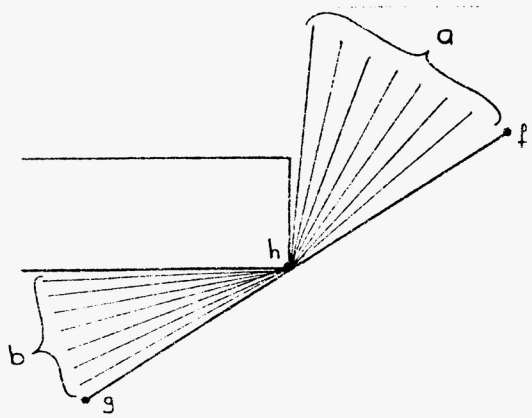

Figure 14