# Efficient Algorithms for the Constraint Generation for Integrated Circuit Layout Compaction

Thomas Lengauer

A 83/06

Fachbereich 10
Universität des Saarlandes
D-6600 Saarbrücken
West Germany

Abstract: A compacter for VLSI layouts is an essential component in many CAD systems for VLSI design. It reduces the area of a given layout violating any of the design rules dictated by the fabrication process.

In many CAD systems for VLSI design the compacter generates a number of linear inequalities from the circuit layout. These so-called constraints restrict the coordinates of the layout components. The resulting inequality system is then solved in some optimum way. The solution of such inequality systems can be done efficiently. The generation of the constraints, however, is a problem for which no efficient algorithms have been devised so far.

We define the graph problem underlying the constraint generation for VLSI circuit compaction. Furthermore we develop efficient, i.e., O(nlogn) time algorithms for the generation of constraint systems that allow to change the layout topology during the compaction in order to yield good compaction results, but at the same time are sparse enough to be solved efficiently, i.e., of size O(n). These algorithms are simple enough to be implemented.

A short version of this report appeared in: Proceedings of the 9th Workshop on Graphtheoretic Concepts in Computer Science (WG'83), June 16-18, 1983, Osnabrück, West-Germany.

## 1. Introduction

Many CAD systems for layout design contain compacters that reduce the layout area using the following strategy [CABBAGE, FLOSS, HILL, MULGA, SLIM, STICKS, TRICKY]. The compaction is done in a number of phases $p_1,\ldots,p_k$. During the odd numbered phases the extent of the layout in the x-direction is reduced by applying a "squeeze" to the layout. The y-coordinates of the layout components are not changed. Analogously the even numbered phases squeeze the layout in the y-direction. Often only two phases exist [CABBAGE,FLOSS]. Also, efforts are made to overcome the independence of the phases by analyzing tradeoffs between them [SLIM,GW82]. We will only be concerned with what happens inside a phase, and not with the interrelationship between phases.

Within a phase there are two approaches to compaction. In one approach compression ridges are run through the layout that mark areas of the layout containing excess space. This space is then removed. This process is iterated until no more compression ridge is found [MULGA,SLIM]. This paper is concerned with the second approach which is graph-theoretic in nature [CABBAGE, FLOSS,HILL,STICKS,TRICKY,GW82]. Let us from now on only consider compaction in x-direction. Constraints are generated between the x-coordinates of the layout components. These constraints take the form

$$x_i - x_j \geq a_{ij} \quad \text{or} \quad x_i = x_j \qquad (a_{ij} \in Z)$$

where $x_i$ and $x_j$ are the coordinates of two components. The constraints can be grouped into three classes.

Type I: Constraints of the form $x_i = x_j$ encode contact rules, i.e. they "solder" layout components together that should contact each other. (Sometimes these constraints take the form $|x_i - x_j| \leq a_{ij} > 0$.)

Type II: Constraints of the form $x_i - x_j \geq a_{ij} > 0$ encode the "design rules" [MC80]. They ensure the minimum separation requirements between different layout components that are dictated by the fabrication process.

Type III: Constraints of the form $x_i - x_j \geq a_{ij} < 0$ encode maximum distance requirements given by the designer explicitly. These constraints ensure circuit performance (by limiting the length of high-capacity wires) or adhere to top-down design (by conforming to floor planning decisions made earlier.

The resulting inequality system is then solved using graph-theoretic methods [L82]. We can assume that in most practical cases the solution time will be $O(m+n)$ where m is the number of inequalities and n is the number of layout components. Therefore we are interested in generating constraint systems that are sparse ($m=O(n)$).

Furthermore the constraint generation should take little time. There are $O(n)$ Type I constraints, and they can easily be generated from a net list accompanying the layout. (If no net list accompanies the layout it can be generated efficiently with a circuit extractor [SW 83].) Type III constraints are given by the designer explicitly. Therefore he has the responsibility over this part of the constraint system. The generation of the Type II constraints is the non-trivial part of the whole constraint generation process, and this paper will focus on it.

## 2. The interval graph

A straightforward way to generate the Type II constraints would be to look at each pair of layout components in turn and use the design rule table to generate the appropriate inequality. However, this would result in $O(n^2)$ inequalities, far too many to be practical.

In fact, many of these inequalities are redundant. Mostly, minimum distances are small (a few microns) such that layout components that are far apart from each other will be assured sufficient separation by the constraint that already exist in each of their neighbourhoods. Therefore only a few of the constraints are actually necessary. We will discuss how to generate such "minimal" constraint systems. To this end we formulate the following graph-theoretic problem.

<u>Definition</u> 1: a) Let L be a set of n vertical intervals in the plane. Each interval is a triple $(x, y_\ell, y_h)$ where x is the x-coordinate and $y_\ell$ and $y_h$ are the y-coordinates of the low and high endpoints of the interval.

b) Let $(L,<)$ be the total ordering that orders L w.r.t. the x-coordinate of each interval. Ties are broken arbitrarily but fixed.

c) Intervals $I_1$ and $I_2$ are said to "overlap", if

$$I_1 \lambda I_2 :\Leftrightarrow I_1 < I_2 \wedge y_{\ell,1} < y_{h,2} \wedge y_{\ell,2} < y_{h,1}$$

d) The following set is called the set of intervals "between" $I_1$ and $I_2$:

$$\beta(I_1, I_2) := \{I \in L;\ I_1 < I < I_2 \wedge I_1 \lambda I \lambda I_2\}.$$

L represents the layout geometry. Specifically, each interval represents a layout component. For a layout to be correct w.r.t. a set of design rules we assume that overlapping intervals have to be separated by certain minimum distances in the x-direction. The exact amounts are of no concern here, since here we are only concerned with the question which constraints have to be generated. (The values in the inequalities can be computed with a simple design rule test.) Non-overlapping intervals are assumed to be sufficiently separated in the y-direction, such that no constraint in the x-direction has to be generated. This can always be ensured by slightly enlarging the interval.

We will define several ways for L to induce a so-called constraint graph G=(L,E). G is a directed acyclic graph with each edge $e=(I_1,I_2)$ representing an inequality of the form $x_2-x_1 \geq a_{21} > 0$. Here $x_1$ and $x_2$ are the x-coordinates of $I_1$ and $I_2$ in the compacted layout.

The constraint graph G is exactly the graph analyzed in [L82]. The source node s that is adjoined to the graph there can be represented by an interval

$$I = (-B,-B,B)$$

where B is a sufficiently large value.

The redundancy of some constraints can now be formulated as the following axiom.

PROCESS AXIOM: If G=(L,E) represents an inequality system that ensures all design rules to be met - we call such a system "admissible" - then the transitive reduction ρ of G also represents such a system.

The process axiom allows us to neglect all inequalities that form "short-cuts" in the constraint graph. This is a realistic assumption because design rules are typically of a highly local nature. The process axiom is a powerful and also essentially the only existing tool for reducing the size of the set of constraints for compaction.

Clearly there are many ways of extracting a constraint graph from layout L. Here is the simplest one.

Definition 2: Let $G_0 = G_0(L) = (L, E_0)$ be defined as follows:

$$(I_1, I_2) \in E_0 \quad \text{iff } I_1 \lambda I_2.$$

Clearly the undirected graph underlying $G_0$ is an interval graph [G80], and its interval representation is given by L, if all x-coordinates are set to zero. Thus the question, how to compute the transitive reduction $\rho_0$ of $G_0$ asks for algorithms to efficiently compute the transitive reduction of such interval dags.

By the remarks above $\rho_0$ is an admissible constraint system. Obviously the following is true:

Fact 1: $(I_1, I_2) \in \rho_0 \Leftrightarrow I_1 \lambda I_2 \wedge B(I_1, I_2) = \emptyset$

Using this characterization $\rho_0$ can be computed in time $O(n \log n)$ with the following algorithm $Gen_0$. $Gen_0$ uses a top-down plane sweep. Thus the algorithm scans a horizontal sweep line across the layout L from top to bottom. During the sweep a data structure D is maintained that stores information about all intervals that currently intersect the sweepline. D is a leaf-chained balanced tree that keeps the intervals in sorted order according to (L,<).

Furthermore with each interval $I \in D$ two pointers to other intervals in L are associated. The pointer left(I) points to nil or an interval less than I. The pointer right(I) points to nil or an interval greater than I. Both pointers represent edges between I and the interval pointed to that are candidates for $\rho_0$ but whose membership in $\rho_0$ has not been decided yet. W.l.o.g. we assume that the y-coordinates for all $I \in L$ are pairwise distinct. Then during the sweep we encounter two kinds of events the algorithm has to deal with, namely the insertion of an interval into the sweep line and the deletion of an interval from the sweep line. Upon these the algorithm does the following:

```
Insert(I):  Insert I into D;
            Find the left and right neighbour I_l and I_r of
            I in D;
            left(I):=I_l; right(I):=I_r;
            left(I_r):=right(I_l):=I;

Delete(I):  if left(I) ≠ nil and left(I)∈D then
              append (left(I),I) to ρ_0;
            if right(I) ≠ nil and right(I)∈D then
              append (I,right(I)) to ρ_0;
```

Theorem 1: a) $Gen_0$ computes $\rho_0$ in time $O(n \log n)$

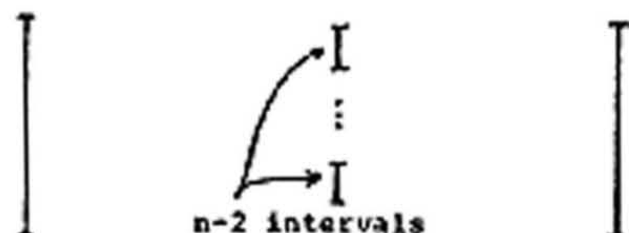
b) $|\rho_0| \le \begin{cases} 0 & \text{if } n=1 \\ 1 & \text{if } n=2 \\ 2n-4 & \text{if } n>2 \end{cases}$

Proof: a) can be shown using a simple induction on the number of events happened during the algorithm. Consider the t-th event. The invariant of the induction is: The edges output so far plus the edges represented by the pointers in D that point to intervals in the sweep-line make up the transitive reduction of $G_0$ $(L_t)$ where $L_t$

is the layout that consists of all parts of the intervals of L that lie above the sweep line.
b) We realize that during each insertion only 2 candidates for $\rho_0$ are produced. Furthermore during the first insertion no candidates for $\rho_0$ are produced and during the second and third insertion only one additional candidate for $\rho_0$ is produced.

□

The upper bound of Theorem 1b is tight, as the following layout shows :




Several systems attempt to find $\rho_0$ [CABBAGE,SLIM]. CABBAGE may produce as many as $O(n^{1.5})$ constraints and typically produces about $O(n^{1.2})$. SLIM comes closer, but it also produces more than the transitive closure, since a constraint is generated between each pair of intervals that are visible from each other at least in part.

## 3. The layer approach

While, given the process axiom, $\rho_0$ is always an admissible constraint system for compaction it is in general not the best one. It does not allow to change the layout topology during compaction, because there is a constraint between an interval and all of its neighbours. [CABBAGE] states this problem without offering a solution. Within our framework we are able to generate different admissible constraint systems that entail more topological freedom.

To this end we define a symmetric and reflexive binary compatibility relation $\pi \subseteq L \times L$. We call two intervals $I_1$ and $I_2$ compatible if $I_1 \pi I_2$. Intuitively two intervals should be compatible if the associated components do not have to meet any minimum distance constraint. Obviously no edge has to exist in the constraint graph between any pair of compatible intervals. Therefore we define:

<u>Definition</u> 3: Let $G_\pi = G_\pi(L) = (L, E_\pi)$ be defined as follows:

$$(I_1, I_2) \in E_\pi :\Leftrightarrow I_1 \lambda I_2 \wedge \neg I_1 \pi I_2$$

We make the reasonable assumption that it can be decided in time $O(1)$ if $I_1 \pi I_2$. One possibility to define $\pi$ is to realize that VLSI circuits are typically laid out on several, say $\ell$ layers that are insulated from each other, except for contact holes hat provide connections between the layers. Therefore we can define: $I_1 \pi I_2$ if the layout components associated with $I_1$ and $I_2$ exist on different layers that are insulated from each other. The resulting graph $G_\pi$ is in general no interval dag and its transitive reduction $\rho_\pi$ may be hard to compute. But we can efficiently compute a supergraph of $\rho_\pi$ with few edges.

<u>Theorem</u> 2: Let $L$ be a layout such that a subset $M(I)$ of a set $\{1,\dots,\ell\}$ of layers is associated with each interval $I \in L$. (We say that $I$ exists on the layers in $M(I)$.) Assume that $|M(I)| \le d$ for all $I \in L$. Let $I_1 * I_2$ iff $M(I_1) \cap M(I_2) = \emptyset$. Then we can in time $O(dn \log n)$ compute a graph $R$ such that $\rho_\pi \subseteq R \subseteq G_\pi$ and $|R| \le 2dn-4$.

<u>Proof</u>: From $L$ we generate $\ell$ layouts $L_1,\dots,L_\ell$. $L_i := \{I \in L \mid i \in M(I)\}$. Then the set of the graphs $G_O(L_i)$ for $i=1,\dots,\ell$ covers all edges of $G_\pi$. Thus the union of all graphs $\rho_\pi(L_i)$ is a supergraph $R$ of $\rho_\pi$. Furthermore $G_O(L_i)$ is a subgraph of $G_\pi$ for $i=1,\dots,\ell$ thus $R \subseteq G_\pi$.
By Theorem 1 the number of edges in $R$ is

$$|R| \le \sum_{i=1}^{\ell} |\rho_O(L_i)| \le 2dn-4 .$$

Furthermore, the time to compute $R$ is $O(\sum_{i=1}^{\ell} |L_i| \log |L_i|) = O(dn \log n)$.

□

[FLOSS] applies this kind of layer separation to achieve some topological freedom during compaction.

## 4. Switching the positions of components within a layer

While $\rho_s$ provides for more topological flexibility by handling each layer of the circuit separately there are still desirable transformations that it does not allow. We give two examples:

Example 1: Jog-flipping of wires :

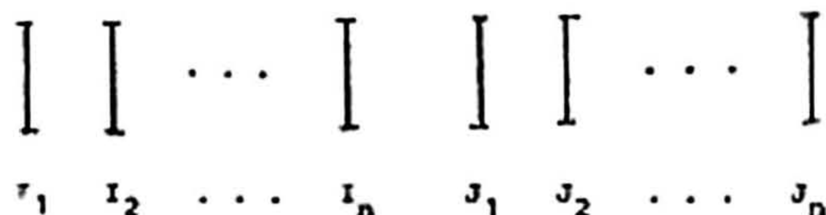
Here the intervals overlap, although slightly. Since they are on the same layer they are incompatible with respect to the above relation ≡ and cannot exchange their positions during compaction. CABBAGE solves this problem by adjusting specifically for such jog flips, the lengths of the intervals temporarily such that they do not overlap. This solution is adhoc, however, and it does not solve the following problem.

Example 2: Transistor flipping :

Here the vertical bold wire is on a top layer and all other structures are on the bottom layer. The contact c connects between the two layers. In this case a simple-minded adjustment of interval lengths will not do. Therefore

we extend $\approx$ by also allowing $I_1 \approx I_2$ if $I_1$ and $I_2$ exist on the same layer and carry the same electrical signal. Such an extension is desirable, since the above example transformations will reduce the area of many layouts significantly. But now $\rho_\approx$ can become large:

Example 3:




Let $I_i \approx I_j$ and $J_i \approx J_j$ for $1 \le i < j \le n$, but $\neg I_i \approx J_j$ for $1 \le i,j \le n$. Then $\rho_\approx$ is the complete bipartite graph $K_{n,n}$, with all edges directed from the left to the right part.

In this case one can use a trick to encode $\rho_\approx$ in small space: One adds a "virtual" interval $K$ between $I_n$ and $I_1$ such that $\neg K \approx I_i$ and $\neg K \approx J_j$, for all $1 \le i,j \le n$. The only purpose of $K$ is to modify $\rho_\approx$ such that it has few edges. However, if we just restrict $\approx$ to being symmetric and reflexive this is not always possible.

Lemma 1: There is a compatibility relation $\approx$ such that $\rho_\approx$ takes $\Omega(n^2)$ space to be encoded.

Proof: We realize that we can induce each $(n,n)$-directed bipartite graph as $G_\approx$ for a layout $L$ with $2n$ intervals. The layout is the same as in Example 3 and $\approx$ is defined such that $I_i \approx I_j$ and $J_i \approx J_j$ for $1 \le i < j \le n$ and $I_i \approx J_j$ if no edge exists between the respective vertices in the bipartite graph. Thus we must encode all $(n,n)$-bipartite graphs. But there are $2^{n^2}/(n!)^2$ such graphs.

(To see this number the vertices in both parts of the graph from 1 to n. There are $(n!)^2$ such numberings, and obviously there are $2^{n^2}$ bipartite graphs that are numbered in this way.) Thus we need at least $\log(2^{n^2}/(n!)^2) = \Omega(n^2)$ bits to encode all such graphs.

□

Lemma 1 shows, that $G_\pi$ can in general not be represented in small space, and thus it is not the right constraint graph for our purposes. We therefore define another constraint graph $G_1$ such that $G_0 \Rightarrow G_1 \Rightarrow G_\pi$ and $G_1$ allows the transformations discussed above.

<u>Definition</u> 4: Let $G_1 = G_1(L) = (L, E)$ be defined as follows:

$$(I_1, I_2) \in E_1 \; :\Leftrightarrow \; I_1 \lambda I_2 \wedge [\neg I_1 \approx I_2 \wedge B(I_1, I_2) \neq \emptyset]$$

Thus $E_1$ can be obtained from $E_0$ by deleting all edges in $\rho_0$ that connect compatible intervals. This allows only exchanges of the positions of neighbouring elements during conpaction. However, both example transformations are included. The transitive reduction $\rho_1$ of $G_1$ can be characterized as follows :

<u>Lemma</u> 2: Define $(I_1, I_2) \in \rho_0^\pi \; :\Leftrightarrow \; (I_1, I_2) \in \rho_0 \wedge I_1 \approx I_2$.
Then for all $I_1$, $I_2$ with $I_1 \lambda I_2$ we have:

$$(I_1, I_2) \in \rho_1 \; :\Leftrightarrow \; (I_1, I_2) \in \rho_0^\pi \wedge \forall I \in B(I_1, I_2): \; (I_1, I) \in \rho_0^\pi \vee (I, I_2) \in \rho_0^\pi$$

<u>Proof</u>: "=>" Indirect: Clearly $(I_1,I_2)\in\rho_0^{\pi}$ implies $(I_1,I_2)\in E_1$. Assume that there is some $I\in B(I_1,I_2)$ such that $(I_1,I)\in\rho_0^{\pi}$ and $(I,I_2)\in\rho_0^{\tau}$. Then $(I_1,I),(I,I_2)\in E_1$, thus $(I_1,I_2)\notin\rho_1$.

"<=": Indirect: Assume $(I_1,I_2)\notin\rho_1$. Then either $(I_1,I_2)\notin E_1$ or there is a path of length $\geq 2$ from $I_1$ to $I_2$ in $E_1$. In the first case $(I_1,I_2)\notin\rho_0^{\pi}$. In the second case there must be an interval $I\in B(I_1,I_2)$ with $(I_1,I)(I,I_2)\in E_1$. Thus $(I_1,I)\in\rho_0^{\pi}$ and $(I,I_2)\in\rho_0^{\tau}$.

□

Lemma 2 provides the basis for an efficient algorithm for computing $\rho_1$. The following lemma shows that information has to be updated only locally during the algorithm.

<u>Lemma 3</u>: Consider an arbitrary position of a horizontal line through the layout L that does not touch the endpoints of any interval in L. Let $\rho_{1,t}$ be the subgraph that is induced from $\rho_1$ by all intervals intersecting the line.

a) If $(I_1,I_2)\in\rho_{1,t}$ then there are at most two intervals $I,I'$ intersecting the line such that $I_1<I<I'<I_2$.

b) The maximum in- and out-degree of any vertex in $\rho_{1,t}$ is 2.

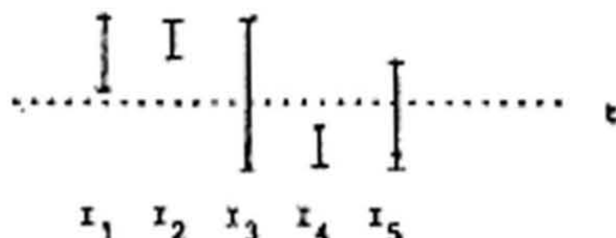
<u>Proof</u>: a) Assume $(I_1,I_2)\in\rho_{1,t}$ and $I_1<I<I'<I''<I_2$ with all intervals intersecting the horizontal line. By definition of $G_1$ $I_1,I',I_2$ is a path in $G_1$ which is a contradiction.

b) We only consider the out-degree. By a) the out-degree of a vertex can be at most 3. If there is an edge from interval $I_1$ to all of its three closest neighbours $I,I',I''$ in the line then $(I_1,I'')\in\rho_1$ implies $(I_1,I)\in\rho_0^*$ by Lemma 2 which contradicts $(I_1,I)\in\rho_1$.

$\square$

The following example shows that we cannot hope to find a simple one-pass algorithm for computing $\rho_1$ using plane sweep methods.

<u>Example</u> 4:




Here $\gamma$ is the equivalence relation with classes $\{I_1,I_3,I_5\}$ and $\{I_2,I_4\}$. At the indicated position of the sweep line it cannot be decided yet whether $(I_1,I_5)\in\rho_1$, but $I_1$ has left the sweep line. Thus any one-pass algorithm would have to entail a certain amount of memory about intervals that have already left the sweep line.

The following algorithm $Gen_1$ is a two-pass algorithm. Here is its description :

<u>Pass</u> 1: Run algorithm $Gen_0$ on L. However, output only edges $(I_1,I_2)\in\rho_0$ such that $I_1 \neq I_2$. Organize the edges in a linear list $\Sigma$ of sets, each set being the collection of edges output during one delete operation.

<u>Pass</u> 2: Make a plane sweep bottom-up. Again maintain a balanced tree D, however this time allow for two pointers to the left and two pointers to the right of each interval to store candidate edges. Furthermore allow for one $\rho_0$-pointer to the left and right to store edges from $\Sigma$.

Insert(I) : Insert I into D;
Fetch from the back of $\Sigma$ all edges that have been output upon the deletion of I in Pass 1, and store them in the $\rho_0$-pointers. Maintain the set of candidate edges between the intervals at most 3 to the right or left of I in D according to Lemma 2, i.e., delete a candidate edge if the newly fetched edges from $\Sigma$ show by Lemma 2 that the edge is not a candidate any more;

Delete(I) : Output all candidate edges that have I as an endpoint and an interval crossing the sweep line as the other;
Delete I from D;

<u>Theorem</u> 3: a) $Gen_1$ produces $\rho_1$ in time $O(n \log n)$

b) $|\rho_1| \leq 4n$

<u>Proof</u>: a) Again we induct on the position t of the sweep line in Pass 2. The invariant is this time: The edges output so far plus the edges stored in the candidate pointers of D form a set $\rho_{1,t}$ of edges such that

for intervals $I_1, I_2$ with $I_1 \lambda I_2$:

$$I_1 \rho_{1,t} I_2 \iff (I_1, I_2) \in \rho_0^{*} \quad \wedge$$

$$\forall I' \in B(I_1', I_2'): \quad (I_1, I) \in \rho_0^{\pi} \vee (I, I_2) \in \rho_0^{\pi}.$$

Here I' is the part of I that lies below the sweep line.

Obviously this invariant is kept by the algorithm, and for $t=\infty$ we get $I'=I$ for all $I \in L$ and thus $\rho_{1,\infty} = \rho_1$ from Lemma 1.

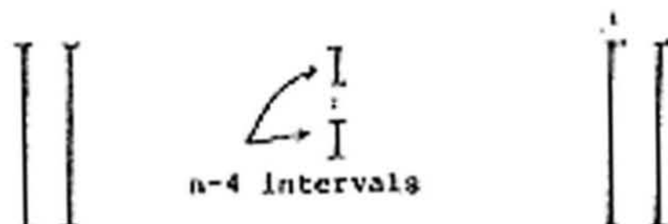
Furthermore note that two candidate edge pointers to the left and right for each interval in D suffice. This is because the proof of Lemma 3b carries over to $\rho_{1,t}$.

b) At each deletion in Pass 2 at most four edges are output by Lemma 3b.

The following example shows that there are layouts such that $|\rho_1| \leq 4n-16$

□

<u>Example</u> 5:



Here $\pi$ is an equivalence relation with the long and the short intervals each forming one equivalence class.

The storage requirement of both algorithms $Gen_0$ and $Gen_1$ is $O(n)$ where m is the maximum number of intervals intersecting the sweep line at any time. Since layouts can be expected to be roughly quadratic with uniform distribution of the layout components we can expect $m=O(\sqrt{n})$. Here we assume that the output of Pass 1 in algorithm $Gen_1$ is on a sequential access storage device and not in main memory. (Otherwise the storage requirement would be linear.) Both algorithms are simple enough to expect that they perform well in practice.

Additional passes can be made across the layout to generate the transitive reductions $\rho_i$ of constraint graphs $G_i=(L,E_i)$ where

$$(I_1,I_2)\in E_i \iff I_1 \lambda I_2 \wedge (\neg I_1 \pi I_2) \vee \exists I \in B(I_1,I_2): (I_1,I),(I,I_2) \in E_{i-1})$$

Then $G_0 \supset G_1 \supset G_2 \supset \ldots$ . After at most n iterations the sequence stabilizes in a graph $G_n$ with the following properties

$$(I_1,I_2)\in E_n \iff I_1 \lambda I_2 \wedge (I_1,I_2) \in G_\pi{}^T.$$

Here $G_\pi$ is the transitive closure of $G_\pi$. Thus $\rho_n=\rho_\pi$. Unfortunately the passes to compute $\rho_i$ become increasingly complex such that this is not a good way to compute $\rho_n$.

## 5. Conclusion

Putting the problem which constraints to generate for compaction into the context of graph theory has enabled us to design efficient algorithms that generate admissible constraint systems allowing up to now unachieved flexibility. Although within a layer topological changes are still confined to pairs of neighbouring components many of the desirable topological transformations are now possible. It may be that transformations of a more global nature can be included in this framework if the compatibility relation is further restricted, say, to being an equivalence relation. Furthermore it is an interesting question to ask, in what way the compaction is improved by iterating $Gen_1$ several times (which is different from adding more passes to $Gen_1$.)

In general it seems that graph theory is a powerful tool for systematizing the algorithms needed in a compacter.

Algorithm $Gen_1$ will be implemented as part of the compacter of the HILL system.

## 6. Acknowledgements

I am indebted to Kurt Mehlhorn for many inspiring discussions on this research. I. Cnop and R. Reischuk independently suggested the top-down direction for a plane sweep. Helmut Alt and Jürgen Doenhardt gave helpful comments on the paper.

## 7. References

[CABBAGE] M.Y.Hsueh, "Symbolic Layout and Compaction of Integrated Circuits", Ph.D.-Thesis, EECS Division, University of California, Berkeley, CA (1979).

[FLOSS] R.A.Auerbach, B.W.Lin, E.A.Elsayed, "Layouts for the Design of VLSI Circuits", Computer Aided Design 13,5 (1981), 271-276.

[HILL] T.Lengauer, K.Mehlhorn, Report on the HILL Specification Language", TR A83/05, Fachbereich Informatik, Univ. d. Saarlandes, Saarbrücken (1983).

[GW82] G.Keden, H.Watanabe, "Optimization Techniques for IC Layout and Compaction", TR 117, Dep. of Comp. Sci., University of Rochester, Rochester, N.Y.,(1982).

[G80] M.C.Golumbic, "Algorithmic Graph Theory and Perfect Graphs", Associated Press (1980).

[L82] T.Lengauer, "On the Solution of Inequality Systems Relevant to IC Layout", Proc. of the 8th Workshop on Graphtheoretic Methods in Comp. Sci. (WG82), Hanser Verlag, München (1982).

[MC80] C.Mead, L.Conway, "Introduction to VLSI Systems", Addision-Wesley (1980).

[MULGA] N.H.E.Weste, "MULGA - An Interactive Symbolic Layout System for the Design of Integrated Circuits", The Bell System Technical Journal 60, (1981) 832-857.

[SLIM] A.E.Dunlop, "SLIM - The Translation of Symbolic Layouts into Mask Data", Proc. of the 17th Design Automation Conference, IEEE (1980) 595-602.

[STICKS] J.D.Williams, "STICKS - A Graphical Compiler for High Level LSI Design", Nat. Comp. Conf. (1978) 289-295.

[SW] T.G.Szymanski, C.J.van Wyk, "Space Efficient Algorithms for VLSI Artwork Analysis", Proc. of the 20th Design Automation Conference, IEEE (1983).

[TRICKY] A.Hanczakowski, "TRICKY-Symbolic Layout System for Integrated Circuits", VLSI Spring Compcon (1981), 374-376.