Linear Constraint Graph for Floorplan Optimization with Soft Blocks*

Jia Wang
Electrical and Computer Engineering
Illinois Institute of Technology
Chicago, IL 60616

Hai Zhou
Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208

***Abstract*— In this paper, we propose the *Linear Constraint Graph* (LCG) as an efficient general floorplan representation. For n blocks, an LCG has at most $2n+3$ vertices and at most $6n+2$ edges. Operations with direct geometric meanings are developed to perturb the LCGs. We apply the LCGs to the floorplan optimization with soft blocks to leverage its advantage in terms of the sizes of the graphs, which will improve the efficiency of solving a complex mathematical program in the inner loop of the optimization that decide the block shapes without introducing overlaps to the non-slicing floorplans. Experimental results confirm that the LCGs are effective and efficient.**

I. Introduction

Floorplanning is an important stage in VLSI circuit design that determines the locations and the shapes of the modules of the circuit on a chip in order to optimize the system subjecting to various objectives and constraints. Many floorplan representations have been proposed for optimizations with simulated annealing (SA). Among the methods that convert from a representation to its physical floorplan, packing, i.e. to push modules to one of the four corners of the chip, is widely used because of its simplicity and its ability to generate area optimal floorplans when the shapes of the modules are fixed. However, if there are soft blocks among the modules whose shapes should be optimized, or the objective of the floorplan optimization is beyond the area minimization, or there are additional placement constraints, packing will not result in optimal floorplans. It is favorable in those cases to explore the solution space of all the non-overlapping floorplans under a given topology in a mathematical programming formulation using a constraint graph to avoid block overlap [1], [2], [3], [4], [5]. It is critical to solve the optimization problem for each topology generated in SA efficiently. From the aspect of the topologies, the corresponding constraint graphs should have small sizes, i.e., to have as few vertices and edges as possible.

Constraint graphs have been studied since the early years of the floorplan research. Polar graphs, which describe the geometric relations between rooms and the maximal line segments for rectangular dissections, were introduced by Ohtsuki et al. [6] and were reviewed by Otten [7]. Although there is no method to perturb the polar graphs in SA, they can be explored by deriving them from mosaic floorplan representations, e.g. Twin Binary Sequences (TBS) [8]. For the general floorplan problem of the block placement, Transitive Closure Graph (TCG) [9] and Adjacent Constraint Graph (ACG) [10], [11] were proposed as the constraint graph based floorplan representations. As TCG keeps all the transitive edges in the graph, the number of the edges in a TCG is $\Theta(n^2)$ for n blocks. In ACG, the sizes of the constraint graphs are reduced intentionally by forbidding over-specifications, transitive edges, and the "crosses", which are special geometric relations that may result in $\Theta(n^2)$ edges in a constraint graph. However, there is no proof showing that the number of the edges in an ACG would be $O(n)$. On the other hand, previous works [1], [2], [3], [4], [5] employed simpler approaches that generate the constraint graphs using the pair-wise geometric relations from either the sequence-pairs [12] or the physical floorplans. The number of the edges in the constraint graphs generated by these approaches might be $\Theta(n^2)$ in the worse case and is $O(n \log n)$ [13] in the average case if the transitive edges are removed.

Our contribution in this paper is to present a class of constraint graphs named *Linear Constraint Graph*s (LCG). LCG is the first general floorplan representation based on the constraint graphs where the numbers of the vertices and the edges are linear to the number of the blocks, which improves upon the previous super-linear size bounds in TCG and ACG. Intuitively, LCGs can be viewed as a combination of the polar graphs and ACGs: the "crosses" are avoided in one dimension as in ACG and are avoided in the other dimension by introducing "bars", which are similar to the maximal line segments in the polar graphs. For n blocks, an LCG contains at most $2n + 3$ vertices and $6n + 2$ edges. We construct an LCG by constructing its horizontal constraint graph first as a *Horizontal Adjacency Graph* (HAG). Generally speaking, HAG captures the horizontal relations between the blocks that are close to each other and is planar. The vertical constraint graph is generated as the *Vertical cOmpanion Graph* (VOG) of the horizontal one, which ensures that every pair of blocks that are not separated horizontally are separated vertically. The operations we designed to perturb the LCGs have direct geometric meaning – such advantage is shared by the constraint graph based representations TCG and ACG. We focus on the application of LCG to the floorplan optimization problems with soft blocks. We emphasize that LCG is preferable when the constraint graphs are essential for the floorplan problem, and LCG is an efficient representation that can be applied to solve general floorplan problems.

The rest of this paper is organized as follows. In Section II, definitions are reviewed. In Section III, we show the motivation of our work. In Section IV, we define LCGs and show its properties. In Section V, we present the operations to perturb LCGs in SA and introduce the framework for floorplan optimization with soft blocks. Experimental results are reported in Section VI. Section VII concludes the paper.

II. Preliminaries

For ease of presentation, we use $V(G)$ and $E(G)$ to denote the set of the vertices and the set of the edges of any graph G respectively.

A constraint graph describes the geometric relations between the blocks in a floorplan. A horizontal directed edge $e = (u, w)$ represents that u is to the left of w while a vertical one represents that u is below w. Four terminal vertices s_h,t_h,s_v,and t_v represent the four boundaries of the floorplan. Any pair of blocks are either separated horizontally or vertically in the constraint graph to avoid overlap. Let B be the set of the rectangular blocks representing the

*The research was conducted at Northwestern University and supported in part by NSF under CNS–0613967.

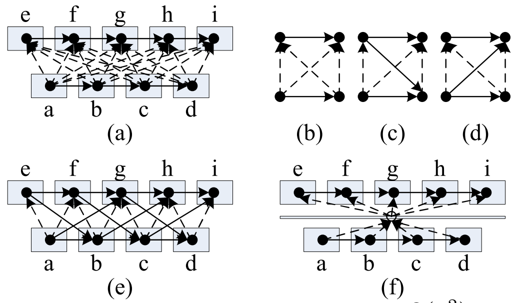

Fig. 1. (a) A floorplan and the constraint graph with $\Theta(n^2)$ edges. (b) A vertical cross as defined by the work of ACG. (c) (d) Two alternatives for the vertical cross. (e) ACG reduces the number of the edges. (f) Alternatively, a bar can be inserted to reduce the number of the edges.

modules in a circuit. We define the constraint graphs as follows formally.

Definition 1 (Constraint Graph): The tuple $G = (C_h, C_v)$ is a constraint graph of the blocks B iff:

CG-1 There are vertices s_h,t_h,s_v,t_v such that $B \cup \{s_h, t_h\} \subseteq V(C_h)$ and $B \cup \{s_v, t_v\} \subseteq V(C_v)$.

CG-2 Both C_h and C_v are directed acyclic graphs (DAG).

CG-3 $\forall u \in B$, there exists a directed path from s_h (and s_v) to t_h (and t_v) in C_h (and in C_v) containing u.

CG-4 For any pair of the blocks, there is a directed path from one of them to the other either in C_h or in C_v.

Let $V(G) = V(C_h) \cup V(C_v)$ and $E(G) = E(C_h) \cup E(C_v)$. The graph C_h and C_v are the horizontal and the vertical constraint graph of G respectively.

Note that we allow dummy vertices besides the terminal vertices and the blocks in the constraint graphs. These dummy vertices are critical for LCG to achieve the linear size.

Recall that B is the set of the blocks. For every $b \in B$, let the width and the height of the block b be $w(b)$ and $h(b)$ respectively. For the dummy vertices besides the terminal vertices and the blocks in $V(G)$, we set $w(b) = h(b) = 0$. The floorplan F with the coordinates of the blocks given as the labelings x and y is *represented* by a constraint graph $G = (C_h, C_v)$ iff we can assign coordinates to the terminal vertices and the dummy vertices such that

$$\begin{aligned} x(i) + w(i) &\leq x(j), \qquad \forall (i,j) \in E(C_h), \\ y(i) + h(i) &\leq y(j), \qquad \forall (i,j) \in E(C_v). \end{aligned} \tag{1}$$

A set of the constraint graphs is called *complete* for a set of blocks, iff for every non-overlapping floorplan of the blocks, there is a constraint graph representing it and belonging to the set.

III. Motivation

An efficient general floorplan representation based on constraint graph is our noble goal, where Adjacent Constraint Graph (ACG) [10], [11] is the first effort. ACG is a constraint graph under three conditions: first, no over-specification, i.e. each pair of blocks are separated either horizontally or vertically but not both; second, no transitive edge, since the corresponding geometric relation is implied; third, no "crosses". A cross is a basic structure in the constraint graph that may result in $\Theta(n^2)$ edges for n blocks, even under the first two conditions. An example is shown in Fig. 1 (a). A vertical cross, which happens many times in (a), is shown in Fig. 1 (b). It simply requires the top two blocks be above the bottom two, making it seems that a horizontal bar is between them. ACG removes such a cross by arguing that removing such a bar

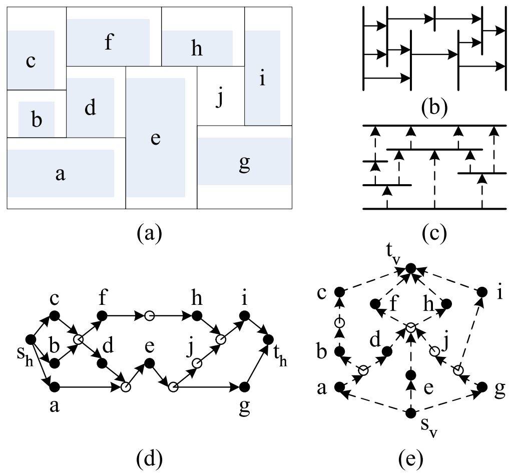

Fig. 2. (a) Convert a block placement to a mosaic floorplan by inserting the dummy room j. (b)(c) The polar graphs. (d)(e) The constraint graph.

cannot worsen the solution – the blocks will fall into one of the two situations as shown in Fig. 1 (c) and (d). The advantage is the reduced number of the edges as shown in Fig. 1 (e). However, it is still open whether the number of the edges in an ACG is bounded by $O(n \log n)$, even though we conjecture so.

Obviously, removing crosses is not the only way to reducing the worse case edge number in a constraint graph. As already mentioned, a cross represents a bar between two pairs of blocks. If a bar of arbitrary length is allowed, we may also be able to reduce edge numbers. For the example shown in Fig. 1 (a), if a horizontal bar, represented by a dummy vertex, is introduced between the two rows of blocks, as shown in Fig. 1 (f), the graph size becomes linear.

Using bars in the floorplan representation is not new, which can be found in polar graphs [6], [7]. The approach is illustrated in Fig. 2. A block placement is first converted to a mosaic floorplan by inserting a dummy room j as shown in Fig. 2 (a). The polar graphs are then constructed in Fig. 2 (b) and (c) where the vertices represent the maximal line segments, i.e. the bars, and the edges represent the rooms. Finally, the horizontal and the vertical constraint graphs shown in Fig. 2 (d) and (e) are derived from the polar graphs by inserting a vertex representing each room on the corresponding edge. However, the drawbacks of the polar graphs include the restriction to the moasic floorplans where dummy rooms are needed for the general floorplans, and the validation difficulty which makes it hard to change a floorplan into another.

Linear Constraint Graph (LCG) achieves the goal by combining ideas from ACG and the polar graph. It forbids horizontal crosses as ACG while introducing horizontal bars as the polar graph. Without dummy vertex, its horizontal graph is similar to ACG but is made planar to break the curse of superlinear edge number. It is defined formally as the *Horizontal Adjacency Graph* (HAG). Its vertical graph is called the *Vertical cOmpanion Graph* (VOG), which is used to separate blocks not separated in HAG where the horizontal bars are introduced to reduce the edge number. The LCG of the floorplan shown in Fig. 2 (a) is given in Fig. 3. This example will be used throughout the paper to illustrate the ideas in LCG.

IV. Linear Constraint Graph

A. *Horizontal Adjacency Graph*

The edges in a horizontal constraint graph can be ordered by the vertical relations among the vertices. Define a DAG to be *ordered*

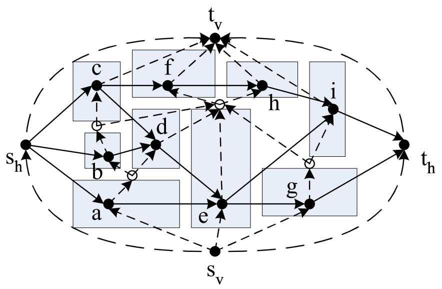

Fig. 3. The LCG of the floorplan in Fig. 2 (a).

if the outgoing edges and the incoming edges of every vertex are ordered. For an ordered DAG G and any vertex $u \in V(G)$, let $R(u)$ and $L(u)$ be the sequences of the ordered outgoing edges and the ordered incoming edges of u respectively. Let $R^+(u)$ and $R^-(u)$ (respectively $L^+(u)$ and $L^-(u)$) be the first and the last element in $R(u)$ (respectively $L(u)$). Let the other vertices besides u of the edges $R^+(u)$, $R^-(u)$, $L^+(u)$, and $L^-(u)$ be $r^+(u)$, $r^-(u)$, $l^+(u)$, and $l^-(u)$ respectively. For a constraint graph, the intuition of using the ordered DAG is to incorporate the vertical order into the horizontal constraint graph, i.e., those two sequences $R(u)$ and $L(u)$ represent the right and the left neighbors of u sorted vertically from bottom to top respectively.

The ordered DAGs are stored in a data structure as follows, which can be treated as "half" of the ACG data structure [10]. Each vertex u maintains two doubly linked lists for $R(u)$ and $L(u)$. Each edge (u, w) stores two vertex pointers pointing to u and w, and four edge pointers pointing to the previous edges and the next edges in $R(u)$ and $L(w)$. Such data structure has the advantage that if a vertex u is given, then $R^+(u)$, $R^-(u)$, $L^+(u)$, and $L^-(u)$ can be accessed in constant time, $R(u)$ and $L(u)$ can be traversed in linear time, and edges can be inserted to or removed from $R(u)$ and $L(u)$ in constant time.

To achieve planarity in the horizontal constraint graph, we define the *above* and the *below* path as follows, which are essentially the boundaries of the faces.

Definition 2 (Above and Below Paths): For any $(u, w) \in E(G)$ and the symbol $\alpha \in \{+, -\}$, the path $P^\alpha(u, w) = (u_1, u_2, \ldots, u_{k+1})$ is the *above* path for $\alpha = +$ or the *below* path for $\alpha = -$ iff:

PAB-1 $\forall 1 \leq i \leq k$, $(u_i, u_{i+1}) \in E(G)$.
PAB-2 $\exists 1 \leq j \leq k$, $u_j = u$ and $u_{j+1} = w$.
PAB-3 $\forall 1 < i \leq k$, $u_{i+1} = r^\alpha(u_i)$; $\forall 1 \leq i < k$, $u_i = l^\alpha(u_{i+1})$.
PAB-4 $L(u_1) = \emptyset$ or $u_2 \neq r^\alpha(u_1)$;
$R(u_{k+1}) = \emptyset$ or $u_k \neq l^\alpha(u_{k+1})$.

For $e = (u, w)$, the above and the below path can be written alternatively as $P^+(e)$ and $P^-(e)$ respectively.

The definition of the above and the below paths can be extended to the vertices that there is at least one edge incident on according to Lemma 1.

Lemma 1: For vertex u, if $L(u) \neq \emptyset$ and $R(u) \neq \emptyset$, then $P^+(L^+(u)) = P^+(R^+(u))$ and $P^-(L^-(u)) = P^-(R^-(u))$. For any such vertex u, if $L(u) \neq \emptyset$, then define $P^+(u) = P^+(L^+(u))$ and $P^-(u) = P^-(L^-(u))$, otherwise define $P^+(u) = P^+(R^+(u))$ and $P^-(u) = P^-(R^-(u))$.

We define the *Horizontal Adjacency Graph* (HAG) as follows. The conditions HAG-1 and HAG-2 are from the requirement of the constraint graph. The condition HAG-3 and HAG-4 ensure that there is no transitive edge in a HAG and are essential for the HAGs to be planar.

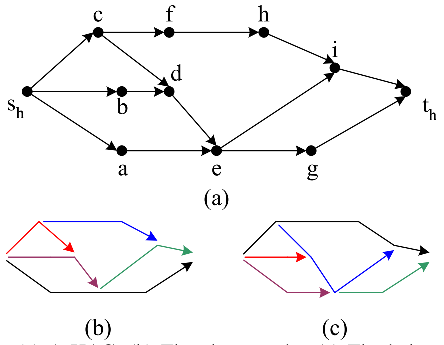

Fig. 4. (a) A HAG. (b) The above paths. (c) The below paths.

Definition 3 (HAG): An ordered DAG C_h is a horizontal adjacency graph of the blocks B iff:

HAG-1 There are vertices s_h and t_h such that $B \cup \{s_h, t_h\} = V(C_h)$.
HAG-2 $\forall u \in B$, there exists a directed path from s_h to t_h in C_h containing u.
HAG-3 $\forall e \in E(C_h)$, both $P^+(e)$ and $P^-(e)$ contain at least two edges.
HAG-4 $\forall u \in V(C_h)$, let $R(u) = (e_1, \ldots, e_d)$. Then $\forall 1 < k \leq d$, there exists $u' \in V(C_h)$ with $L(u') = (e'_1, \ldots, e'_{d'})$ and $1 < j \leq d'$ such that $P^+(e_k) = P^+(e'_j)$ and $P^-(e_{k-1}) = P^-(e'_{j-1})$.

The example of a HAG and all the above and the below paths are shown in Fig. 4. The faces and their boundaries can be identified to understand the intuition behind the above and the below paths. The path pairs referred in the condition HAG-4 are high-lighted with the same color. The exceptions are the top above path and bottom below path. We have the following lemma regarding them.

Lemma 2: $P^+(s_h) = P^+(t_h)$ and $P^-(s_h) = P^-(t_h)$.

Moreover, there is no transitive edge according to Lemma 3.

Lemma 3: In a HAG C_h, if $(u, w) \in E(C_h)$, then there is no path from u to w in C_h longer than one edge.

B. The Top-Insert Lemma

To further explore the properties of a HAG, e.g. the size and the planarity, we present the Top-Insert Lemma that can be used to reason upon HAGs through the mathematical induction.

It is straight-forward that if $|B| = 1$, then there is only one HAG – assuming $B = \{b\}$, we can write the HAG as C_h^1 with $V(C_h^1) = \{s_h, b, t_h\}$ and $E(C_h^1) = \{(s_h, b), (b, t_h)\}$. Given a HAG and a new block, we build a new HAG using the InsertTop subroutine as shown in Fig. 5. Generally speaking, this subroutine constructs a new HAG by inserting a new vertex to the top of an existing one. On line 1, the new block b is inserted to the vertex set. If $E(C_h)$ contains the edge (a, c), it is removed on line 2 to satisfy the condition HAG-3. Two new edges are inserted on line 3 and line 4.

```
Subroutine InsertTop
Inputs
  $C_h$: a HAG. $b$: a new block.
  $a$, $c$: two vertices on $P^-(s_h)$ in $C_h$.
Output Updated $C_h$.
1   Insert $b$ to $V(C_h)$.
2   If $r^-(a) = c$: remove $(a, c)$ from $E(C_h)$.
3   Insert $(a, b)$ to $E(C_h)$ such that $R^-(a) = (a, b)$.
4   Insert $(b, c)$ to $E(C_h)$ such that $L^-(c) = (b, c)$.
```

Fig. 5. The InsertTop Subroutine

The correctness and the time complexity of the subroutine are stated in the following lemma.

Lemma 4: Assume C_h is a HAG of the blocks B, $b \notin B$, a and c are on the below path $P^-(s_h)$ in C_h, and c is after a on $P^-(s_h)$. Then InsertTop updates C_h into a HAG of the blocks $B \cup \{b\}$ in constant time.

For the ease of presentation, define T to be the function that represents the output of the InsertTop subroutine, i.e., let $T(C_h, b, a, c)$ be the updated HAG obtained by InsertTop(C_h, b, a, c). The following Top-Insert Lemma shows that every HAG can be built from C_h^1 using the InsertTop subroutine with proper parameters.

Lemma 5 (The Top-Insert Lemma): Let C_h be a HAG of the blocks B where $|B| > 1$. Then there exist vertices a, b, and c such that, first, a, b, and c are three consecutive vertices on $P^-(s_h)$ in C_h; second, $C_h = T(C'_h, b, a, c)$ for some HAG C'_h of the blocks $B - \{b\}$.

Proof: We claim there is a vertex b on $P^-(s_h)$ such that b has exactly one incoming edge and one outgoing edge. Let $P^-(s_h) = (u_0, \dots, u_{k+1})$ where $k \geq 1$, $u_0 = s_h$, and $u_{k+1} = t_h$. We prove the claim by contradiction, i.e., to assume that, $\forall 1 \leq j \leq k$, u_j has at least two incoming edges or at least two outgoing edges. Based on the assumption, we first prove that, $\forall 1 \leq j \leq k$, u_j has exact one incoming edge and at least two outgoing edges by induction on j. For $j = 1$, the vertex u_1 has no incoming edge other than (u_0, u_1) – otherwise $P^+(u_0, u_1)$ has only one edge, which violates HAG-3. Thus u_1 should have at least two outgoing edges. Suppose u_{j-1} has at least two outgoing edges for $j > 1$. Similar to the argument for the case $j = 1$, we have that u_j has only one incoming edge. Thus u_j should have at least two outgoing edges. Therefore, we proved that, $\forall 1 \leq j \leq k$, u_j has exact one incoming edge and at least two outgoing edges. On the other hand, because of the symmetry, we can also prove that, $\forall 1 \leq j \leq k$, u_j has exact one outgoing edge and at least two incoming edges. A contradiction is reached. Thus the claim holds and such vertex b exists.

Let (a, b) and (b, c) be the incoming and the outgoing edges respectively. Then a, b, and c are three consecutive vertices on $P^-(s_h)$ in C_h. We construct C'_h by first removing the vertex b and the edges (a, b) and (b, c) from C_h. If there is no path from a to c in C'_h, then we modify C'_h by inserting an edge (a, c) such that $r^-(a) = c$ and $l^-(c) = a$. It is straight-forward to verify that, first, C'_h is a HAG of the blocks $B - \{b\}$; second, a and c are both on $P^-(s_h)$ in C'_h and a is before c; third, $C_h = T(G'_h, b, a, c)$. Therefore, we proved the lemma. ■

According to the Top-Insert Lemma, we have,

Corollary 1: Suppose C_h is a HAG of the blocks B, then $|V(C_h)| = |B| + 2$ and $|E(C_h)| \leq 2|B|$.

Lemma 6: Every HAG is planar.

C. Vertical Companion Graph

Once we have a HAG as the horizontal constraint graph, a vertical constraint graph can be constructed accordingly to separate blocks not separated horizontally, with dummy vertices for horizontal bars. We define the *Vertical cOmpanion Graph* (VOG) recursively as follows, which will be the vertical constraint graph. First of all, for the HAG C_h^1 of one block b, let the vertical companion graph be C_v^1 satisfying that $V(C_v^1) = \{s_h, t_h, b, s_v, t_v\}$ and $E(C_v^1) = \{(s_v, s_h), (s_v, b), (s_v, t_h), (s_h, t_v), (b, t_v), (t_h, t_v)\}$. For the HAG C_h of the blocks B where $|B| > 1$, suppose $C_h = T(C'_h, b, a, c)$ for some HAG C'_h of the blocks $B - \{b\}$ according to the Top-Insert Lemma. Assume the VOG of C'_h to be C'_v. We construct the VOG of C_h using the CoInsertTop subroutine as shown in Fig. 6. The intuition is to insert new edges such that for every block that is not separated horizontally with b, there is a path in C_v from the vertex to b. Note that in the subroutine, we assume for every $u \in V(C'_h)$, there are two unique vertices u^+ and u^- in $V(C'_v)$ such that both (u^+, u) and (u, u^-) belong to $E(C'_v)$. The validity of this assumption will be established when we proved the correctness of the subroutine in Lemma 7. On line 1, the vertex b is inserted to the VOG. If (a, c) is an edge in C_h, we insert one new edge to C_v on line 3 and insert another new edge on line 5 or 7 depending on whether $r^+(a) = c$ or $l^+(c) = a$. Otherwise, a dummy vertex f is inserted on line 9 before two edges are inserted on line 10. The end points of the outgoing edges from some of the vertices on $P^-(s_h)$ are replaced in the loop on line 12. There is a path in C_v from a^+ to f (through $r^-(a)$) if $a = l^+(r^-(a))$. Otherwise we insert the edge (a^+, f) on line 14. Similarly, the edge (c^+, f) is inserted on line 15 iff there is no path in C_v from c^+ to f.

```
Subroutine CoInsertTop
Inputs
  $C_h$,$C_v$:a HAG and the VOG of it.
  $b$       :a new block.
  $a$, $c$    :two consecutive vertices on $P^-(s_h)$ in $C_h$.
Output Updated $C_v$.
 1  Insert $b$ to $V(C_v)$.
 2  If $r^-(a) = c$:
 3    Insert $(b, t_v)$ to $E(C_v)$.
 4    If $r^+(a) = c$:
 5      Insert $(a^+, b)$ to $E(C_v)$.
 6    Else:// must have $l^+(c) = a$
 7      Insert $(c^+, b)$ to $E(C_v)$.
 8  Else:
 9    Insert a new vertex $f$ to $V(C_v)$.
10    Insert $(f, b)$ and $(b, t_v)$ to $E(C_v)$.
11    Let $P'$ be the sub-path of $P^-(s_h)$ from $a$ to $c$.
12    For each vertex $u$ on $P'$ except $a$ and $c$:
13      Replace $(u, t_v)$ with $(u, f)$ in $E(C_v)$.
14    If $a \neq l^+(r^-(a))$: insert $(a^+, f)$ to $E(C_v)$.
15    If $c \neq r^+(l^-(c))$: insert $(c^+, f)$ to $E(C_v)$.
```

Fig. 6. The CoInsertTop Subroutine

The CoInsertTop subroutine is consistent because of the following lemma.

Lemma 7: Suppose C_v is the VOG of a HAG C_h. Then, first, $V(C_h) \subset V(C_v)$ and C_v is a DAG. Second, $\forall u \in V(C_h)$, there is exactly one incoming edge (u^+, u) and one outgoing edge (u, u^-) in C_v, and both u^+ and u^- do not belong to $V(C_h)$. Third, for every vertex u on $P^-(s_h)$, $u^- = t_v$, and for every vertex u on $P^+(s_h)$, $u^+ = s_v$. Fourth, if (a, c) is an edge in $P^-(s_h)$ in C_h, then at least one of $r^+(a) = c$ and $l^+(c) = a$ holds. If both of them hold, then $a^+ = c^+$.

An example of the VOG of the HAG shown in Fig. 4 (a) is shown in Fig. 7. We have the following corollary concerning the size of a VOG according to the CoInsertTop subroutine.

Corollary 2: Suppose C_v is a VOG of the HAG of the blocks B, then $|V(C_v)| \leq 2|B| + 3$ and $|E(C_v)| \leq 4|B| + 2$.

D. Linear Constraint Graph

Based on the definition of HAG and VOG, the *Linear Constraint Graph* (LCG) is formally defined as follows.

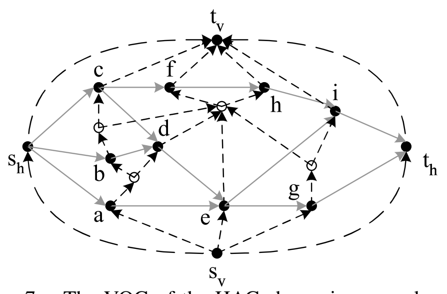

Fig. 7. The VOG of the HAG shown in gray edges.

Definition 4 (LCG): For the blocks B, the linear constraint graph G is a tuple (C_h, C_v), where C_h is a HAG of B and C_v is a VOG of C_h. Let $V(G) = V(C_h) \cup V(C_v)$ and $E(G) = E(C_h) \cup E(C_v)$.

Lemma 8: An LCG is a constraint graph.

Proof: We prove that CG-4 holds for any LCG $G = (C_h, C_v)$ for the blocks B by induction on $|B|$. If $|B| = 1$, obviously CG-4 holds since there is only one vertex in B.

Suppose CG-4 holds for $|B| = n-1$. For $|B| = n$, according to the Top-Insert lemma, assume $C_h = F(G'_h, b, a, c)$. Let G'_v be the VOG of G'_h and let C_v be obtained by the CoInsertTop subroutine. Let $P' = (u_0, \ldots, u_k)$ be the sub-path of $P^-(s_h)$ in G'_h from a to c, i.e. $u_0 = a$ and $u_k = c$. Let $w \in B - \{b\}$. According to Lemma 3, $\forall 0 \le j < k$, there is no path from u_j to u_{j+1} in G'_h. According to Lemma 7, $\forall 0 \le j \le k$, $u_j^- = t_v$. Thus, $\forall 0 \le j \le k$, there is no path from u_j to w in G'_v. Therefore, if we consider the paths between w and u_j, $\forall 0 \le j \le k$, in G'_h and G'_v, there are 4 possible cases as follows. First, there is a path from w to a in G'_h or $w = a$. Then there is a path from w to b in C_h. Second, there is a path from c to w in G'_h or $w = c$. Then there is a path from b to w in C_h. Third, $w = u_j$ for some j satisfying $0 < j < k$. Then there is a path from w to b through b^+ in C_v. Fourth, there is a path from w to u_j in G'_v for some j satisfying $0 \le j \le k$. Recall (u_j^+, u_j) is the only incoming edge of u_j in G'_v. Then the path must contain u_j^+. Thus there is a path from w to b through u_j^+ in C_v. So we proved CG-4 holds for $w \in B - \{b\}$ and $u = b$. It is straight-forward to verify that CG-4 holds for $w, u \in B - \{b\}$ in C_h and C_v according to the induction hypnoses. Therefore, CG-4 holds when $|B| = n$.

It is straight-forward that CG-1, CG-2, and CG-3 hold for an LCG. Since we have shown that CG-4 holds for an LCG, we proved that an LCG is a constraint graph. ■

If a non-overlapping floorplan is given, the FPToLCG algorithm shown in Fig. 8 constructs the LCG representing the floorplan. The algorithm builds the LCG by inserting the blocks according to the ascending order of their y-coordinates. From the LCG constructed on line 3 and 4, the subroutine CoInsertTop and InsertTop are called to insert new blocks on line 9 and 11. The labelings x and y are extended on line 3, 4, and 10. The below path $P^-(s_h)$ in C_h is maintained as P throughout the algorithm on line 5 and 12. It is implemented as an AVL tree or a Red-Black tree [14] that stores the vertices on the path according to their x labelings.

The correctness and the complexity of the FPToLCG algorithm are stated in the following lemma.

Lemma 9: Assume F is a non-overlapping floorplan of the blocks B. The FPToLCG algorithm will terminate and generate an LCG $G = (C_h, C_v)$ that represents F. The time complexity is $O(|B| \log |B|)$ and and space complexity is $O(|B|)$.

In summary, we have the following theorem as the major result of this paper according to Lemma 8 and 9, and Corollary 1 and 2.

```
Algorithm FPToLCG
Inputs A non-overlapping floorplan $F$.
Output The LCG $G = (C_h, C_v)$.
1   Sort the blocks $B$ into $(b_1, b_2, \ldots, b_{|B|})$ according to
    the y-coordinates $y(b), \forall b \in B$.
2   $M \leftarrow \max_{b \in B}\{|x(b)|, |y(b)|, |x(b)+w(b)|, |y(b)+h(b)|\}$.
3   $V(C_h) \leftarrow \{s_h, b_1, t_h\}$, $E(C_h) \leftarrow \{(s_h, b_1), (b_1, t_h)\}$.
    $(x(s_h), x(t_h), w(s_h), w(t_h)) \leftarrow (-M, M, 0, 0)$.
4   $V(C_v) \leftarrow V(C_h) \cup \{s_v, t_v\}$,
    $E(C_v) \leftarrow \bigcup_{u \in V(C_h)} \{(s_v, u), (u, t_v)\}$.
    $(y(s_h), y(t_h), y(s_v), y(t_v)) \leftarrow (0, 0, -M, M)$.
5   The path $P \leftarrow (s_h, b_1, t_h)$.
6   For $i = 2$ to $|B|$:
7     $a \leftarrow \text{argmax}_{\{u: u \in P, x(u)+w(u) \le x(b_i)\}} x(u)$.
8     $c \leftarrow \text{argmin}_{\{u: u \in P, x(b_i)+w(b_i) \le x(u)\}} x(u)$.
9     $C_v \leftarrow$ CoInsertTop$(C_h, C_v, b_i, a, c)$.
10    If $r^-(a) \ne c$: $y(b_i^+) \leftarrow y(b_i)$.
11    $C_h \leftarrow$ InsertTop$(C_h, b_i, a, c)$.
12    Remove all the vertices between $a$ and $c$ in $P$.
      Insert $b_i$ to $P$ between $a$ and $c$.
```

Fig. 8. The FPToLCG Algorithm

Theorem 1: Let the set of all the LCGs of a set of blocks B be $\mathcal{L}_B$. Then $\mathcal{L}_B$ is complete for B and $\forall G \in \mathcal{L}_B$, $|V(G)| \le 2|B|+3$ and $|E(G)| \le 6|B|+2$.

V. LCG Floorplan Optimization

We perform floorplan optimization using LCGs by SA. Two most important issues are addressed in the following two sub-sections: one is to design operations that perturb the representation and the other is to evaluate the representation through a cost function.

A. Perturbations of LCG

Recall that an LCG consists of a HAG and a VOG. We design three operations that perturb the LCGs. The first one is with the name exchange and exchanges two blocks in both the HAG and the VOG. The next two operations are designed to first perturb the HAGs and then update the VOGs accordingly since the VOG can be derived from a HAG. In the operation with the name insertH, an edge is inserted between two vertices in the HAG which changes the vertical relation between them into a horizontal one. In the operation with the name removeH, an edge in the HAG is removed such that the horizontal relation between the end points of the edge is changed to a vertical one. We only present the changes in the HAGs for these two operations while omit the changes in the VOGs for simplicity.

Suppose the LCG is $G = (C_h, C_v)$. The insertH operation is illustrated in Fig. 9. Recall that for $u \in V(C_h)$, (u^+, u) and (u, u^-) are the only edges incident on u in C_v. Two vertices $a \in B$ and $b \in B$ are selected such that $a^+ = b^-$. Such vertices exist when C_h is not a single path from s_h to t_h. We can insert either (b, a) or (a, b) to C_h. Because of the symmetry, assume that the edge (b, a) should be inserted without loss of generality. Let $c = l^+(a)$ and $d = r^-(b)$. The following lemma holds for $P^+(a)$ and $P^-(b)$ before inserting (b, a).

Lemma 10: If for $a, b \in V(C_h)$ we have $a^+ = b^-$, then the end points of $P^+(a)$ and $P^-(b)$ are the same.
If c is the first vertex of $P^+(a)$ and $P^-(b)$, there is a path from c to b in C_h. Thus the edge (c, a) should be removed such that HAG-3 would not be violated after inserting (b, a). Similarly, if d is the last vertex of $P^+(a)$ and $P^-(b)$, the edge (b, d) should be removed.

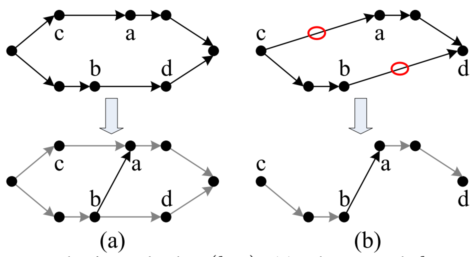

Fig. 9. Insert a horizontal edge (b, a): (a) when c and d are not the end points of $P^+(a)$; (b) when c and d are the end points of $P^+(a)$, (c, a) and (b, d) should be removed.

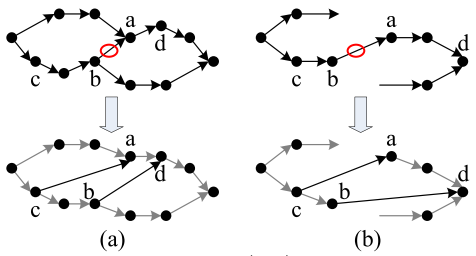

Fig. 10. Remove the horizontal edge (b, a): (a) when a and b have at least 2 incoming and outgoing edges respectively, (c, a) and (b, d) are optional; (b) when a and b have only 1 incoming and outgoing edges respectively, (c, a) and (b, d) must be inserted.

Finally, the edge (b, a) is inserted to C_h such that $r^-(b) = a$ and $l^+(a) = b$.

The removeH operation is illustrated in Fig. 10. An edge $(b, a) \in E(C_h)$ with $a \in B$ and $b \in B$ is selected such that either $r^-(b) = a$ and $l^+(a) = b$, or $r^+(b) = a$ and $l^-(a) = b$. Such edge exists when there is an edge in C_h whose end points both belong to B. Because of the symmetry, assume that $r^-(b) = a$ and $l^+(a) = b$ without loss of generality. If a has more than one incoming edges in C_h, an optional new edge (c, a) can be inserted to C_h where c is a vertex on $P^-(b)$ between the first vertex and b. Otherwise, if a has only one incoming edge, i.e. (b, a), a new edge (c, a) must be inserted where c is either the first vertex of $P^-(b)$ or a vertex on $P^-(b)$ between the first vertex and b. Similarly, if b has more than one outgoing edges, an optional new edge (b, d) can be inserted where d is a vertex on $P^+(a)$ between a and the last vertex; if b has only one outgoing edge, a new edge (b, d) must be inserted where d is either the last vertex of $P^+(a)$ or a vertex on $P^+(a)$ between a and the last vertex. Finally, the edge (b, a) is removed from C_h.

The correctness and the time complexity of the insertH and the removeH operations are stated in the following lemma.

Lemma 11: Both the insertH and the removeH operations change an LCG into another one. The time complexity is $O(n)$ for n blocks.

An insertH operation can be reverted by a removeH operation. Moreover, a dummy vertex in the VOG of any LCG can be removed by the insertH operation as shown in Fig. 9 (b) without introducing new dummy vertices. Thus we would obtain an LCG with no dummy vertex in the VOG from any LCG of n blocks by applying the insertH operation at most $n - 1$ times. The HAGs in such LCGs consist of a single path from s_h to t_h and the conversions among them can be done by only the exchange operation. Thus we can obtain any LCG from such LCGs via reverting the insertH operations by the removeH operations. Therefore, we have the following theorem.

Theorem 2: Applying the three operations exchange, insertH, and removeH can convert any LCG to any other LCG. For n blocks, it takes at most $3n$ operations for such conversions.

TABLE I

RESULTS FOR AREA OPTIMIZATION W/ SOFT BLOCKS.

		SP+TR			LCG+TR		
name	n	ds(%)	t(s)	$\|E\|$	ds(%)	t(s)	$\|E\|$
apte	9	**0.04**	34	40	**0.04**	21	34
xerox	10	**0.08**	43	47	**0.08**	41	38
hp	11	**0.09**	41	54	0.13	26	41
ami33	33	0.28	383	347	**0.24**	179	125
ami49	49	**0.24**	694	679	0.27	319	182

TABLE II

RESULTS FOR WIRE LENGTH OPTIMIZATION W/ SOFT BLOCKS.

	SP+TR			LCG+TR		
name	ds(%)	wl(mm)	t(s)	ds(%)	wl(mm)	t(s)
apte	0.09	125.29	35	**0.08**	**125.28**	26
xerox	0.21	**145.16**	46	**0.15**	152.69	36
hp	0.26	43.42	37	**0.22**	**42.60**	27
ami33	0.50	57.89	349	**0.49**	**52.46**	236
ami49	1.17	290.89	615	**0.64**	**272.23**	442

B. Floorplan Optimization with Soft Blocks

When there are soft blocks, the block shapes should be determined first to evaluate a floorplan. We follow the approach of Lin et al. [3] to formulate a mathematical programming problem using the constraint graph as the constraints and to solve the problem by Lagrangian relaxation in order to obtain optimal block sizes. This approach was previously proposed by Young et al. [1] and was improved in the work [3]. We only introduce the relevant part in this section while the details and the reviews of the previous works should be found in the works [1], [3].

Recall that B is the set of the blocks. Suppose that each block $b \in B$ has an area $A(b)$. Let the decision variables be the block widths $w(b)$ within the predefined ranges. The P$_{\text{peri}}$ problem is formulated to optimize the perimeter of the floorplan bounding box under a given LCG. The constraints are in the form of the inequalities in Eq. (1) while the block heights $h(b)$ are expressed as $\frac{A(b)}{w(b)}$.

It was shown in [3] that P$_{\text{peri}}$ is a convex programming formulation under the variable transformation $\text{logw}(b) = \log w(b)$, and thus can be solved by Lagrangian relaxation that associates each constraint a Lagrangian multiplier to formulate the Lagrangian dual problem where the decision variables are the multipliers. The Lagrangian dual problem can be simplified into the LD$_{\text{peri}}$ problem by constraining the multipliers as a network flow in the constraint graph, given the special structure that the constraints are a system of difference inequalities. As the objective function of the LD$_{\text{peri}}$ problem is not differentiable in general, Lin et al. [3] proposed a trust-region method to solve the LD$_{\text{peri}}$ problem. In each iteration of this method, a min-cost network-flow problem on the constraint graph is formulated based on the current set of multipliers and a step size, and is solved to generate a new set of multipliers. Depending on the improvement of the objective function of the LD$_{\text{peri}}$ problem in comparison to the expected improvement based on the first order approximation, either the new set of multipliers would be rejected and the step size would be decreased, or the new set of multipliers would be accepted and the step size would be increased or remain the same.

VI. EXPERIMENTAL RESULTS

We obtain the code of the floorplanner in the work [3], which was based on the code of the floorplanner in the work [1] and used CS2 version 4.3 [15] as the min-cost network flow solver. We replace the sequence-pair representation in the code with our LCG representation which is implemented in C++. Both the code

TABLE III

RESULTS FOR WIRE LENGTH OPTIMIZATION W/ HARD BLOCKS.

		Parquet			ACG				LCG			
name	n	ds(%)	wl	t(s)	ds(%)	wl	t(s)	$\|E\|$	ds(%)	wl	t(s)	$\|E\|$
n100	100	**7.95**	323594	30	8.33	308181	30	627	8.60	**300469**	27	347
n200	200	10.99	581176	150	10.26	540006	137	1456	**9.03**	**537882**	133	696
n300	300	12.05	709182	288	13.00	643538	357	2248	**11.44**	**638710**	274	1052

of the work [3] and our code are compiled by GCC version 3.4 and executed on a Linux workstation with two 927MHz Pentium III processors and 512MB memory. The same setting of simulated annealing is used in both code.

We follow Lin et al. [3] to setup the experiments. There are 5 benchmarks derived from the MCNC benchmark suite. The aspect ratio of each block is between 0.5 and 2. Area optimization is performed with the cost function being the perimeter of the floorplan bounding box. Wire length optimization is performed with the cost function being the summation of the perimeter and the average half-perimeter wire length (HPWL) of all the nets.

For each of the benchmarks and each optimization, we run each program for 5 times and compare the best results. The results from area optimization and wire length optimization are reported in Table I and II respectively. For each benchmark, the name and the number of the blocks are shown in the columns “name” and “n” respectively. The results from our approach are shown in the columns “LCG+TR”. The results from the approach by Lin et al. [3] are shown in the columns “SP+TR”. For both the area and the wire length optimizations, the deadspace in percentage and the running time in seconds are reported in the columns “ds(%)” and “t(s)” respectively. We observe that more than 95% of the runtime is spent to solve the $\mathrm{LD}_{\mathrm{peri}}$ problem by the trust-region method in both approaches. The average numbers of the edges of the constraint graphs generated in SA are reported in the columns “$|E|$” for the area optimization. These numbers are similar for the wire length optimization and thus are omitted. The total HPWL in millimeter are reported in the columns “wl(mm)” for the wire length optimization.

It can be seen from the tables that although the results of the approach by Lin et al. [3] are already almost optimal, our approach can improve the qualities for 1 benchmark for the area optimization and 4 benchmarks for the wire length optimization in much less time while the qualities for other cases are similar. It is clear that there are always less edges in the constraint graphs in our approach and the gap between the number of edges in our approach and that in the approach by Lin et al. [3] increases as the size of the benchmark increases. Note in the works [1], [3], when the constraint graphs were constructed from the sequence-pairs, the transitive edges were removed. This implies that in a similar approach to maintain TCGs without transitive edges, the average numbers of the edges would be similar to those of the approach by Lin et al. [3], which are more than those of our approach.

It is also interesting to investigate the performance of LCG as a general floorplan representation for hard blocks. We obtain two SA based floorplanners for comparison: the ACG floorplanner [10], [11] and the Parquet floorplanner [16] (version 4.0). We implement our LCG floorplanner by integrating our LCG implementation with the SA optimization framework in the ACG floorplanner. All the floorplanners are compiled and executed under the same system setting as mentioned before. The cost function of SA is the weighted summation of the area and the HPWL with a weight ratio of $1:1$. We adopt the same annealing schedule for the ACG and the LCG floorplanners. The Parquet floorplanner is running in the free-outline mode with the sequence-pair representation and a pre-defined running time limit which is similar to the running time of our LCG floorplanner. We perform experiments with three benchmarks from the GSRC benchmark suite: n100, n200, and n300. For each of the benchmarks, we run each floorplanner for 5 times and compare the best results in Table III. The results from the Parquet floorplanner, the ACG floorplanner, and our LCG floorplanner are shown in the columns “Parquet”, “ACG”, and “LCG” respectively. Each individual column has the same meaning as mentioned before. It can be seen that our LCG floorplanner can generate floorplans with better quality in less time for almost all the cases compared to the Parquet and the ACG floorplanners. Moreover, there are always less edges in the LCGs than in the ACGs.

VII. CONCLUSIONS

In this paper, we proposed the *Linear Constraint Graph*s (LCG) as a general floorplan representation based on the constraint graphs. For n blocks, we showed that each LCG has at most $2n+3$ vertices and at most $6n+2$ edges. We proved that LCGs can represent any non-overlapping floorplans. We designed operations that have direct geometric meaning to perturb LCGs in simulated annealing and proved such perturbations are sufficient to explore all the LCGs stochastically. The advantages of LCGs is confirmed by the experimental results.

REFERENCES

[1] F. Y. Young, C. C. N. Chu, W. S. Luk, and Y. C. Wong, “Handling soft modules in general non-slicing floorplan using Lagrangian relaxation,” *IEEE TCAD*, vol. 20, no. 5, pp. 687–692, May 2001.

[2] E. F. Y. Young, C. C. N. Chu, and M. L. Ho, “Placement constraints in floorplan design,” *IEEE TVLSI*, vol. 12, no. 7, pp. 735–745, July 2004.

[3] C. Lin, H. Zhou, and C. Chu, “A revisit to floorplan optimization by Lagrangian relaxation,” in *ICCAD*, 2006, pp. 164–171.

[4] X. Tang, R. Tian, and M. D. F. Wong, “Minimizing wire length in floorplanning,” *IEEE TCAD*, vol. 25, no. 9, pp. 1744–1753, Sept. 2006.

[5] H.-C. Lee, Y.-W. Chang, and H. H. Yang, “MB*-Tree: A multilevel floorplanner for large-scale building-module design,” *IEEE TCAD*, vol. 26, no. 8, pp. 1430–1444, Aug. 2007.

[6] T. Ohtsuki, N. Sugiyama, and H. Kawanishi, “An optimization technique for integrated circuit layout design,” in *Proc. ICCST*, Kyoto, Japan, 1970, pp. 67–68.

[7] R. H. Otten, “What is floorplan?” in *ISPD*, 2000, pp. 201–206.

[8] F. Y. Young, C. C. N. Chu, and Z. C. Shen, “Twin Binary Sequences: A non-redundant representation for general non-slicing floorplan,” *IEEE TCAD*, vol. 22, no. 4, pp. 457–469, Apr. 2003.

[9] J.-M. Lin and Y.-W. Chang, “TCG: A transitive closure graph-based representation for non-slicing floorplans,” in *DAC*, 2001, pp. 764–769.

[10] H. Zhou and J. Wang, “ACG-adjacent constraint graph for general floorplans,” in *ICCD*, 2004, pp. 572–575.

[11] J. Wang and H. Zhou, “Interconnect estimation without packing via ACG floorplans,” in *ASP-DAC*, 2005, pp. 1152 – 1155.

[12] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, “VLSI module placement based on rectangle-packing by the sequence-pair,” *IEEE TCAD*, vol. 15, no. 12, pp. 1518–1524, Dec. 1996.

[13] C. Lin, “Incremental mixed-signal layout generation concepts,” Ph.D. dissertation, Eindhoven University of Technology, Eindhoven, The Netherlands, 2002.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.

[15] CS2 version 4.3, “Andrew Goldberg’s network optimization library,” http://www.avglab.com/andrew/soft.html.

[16] S. N. Adya and I. L. Markov, “Fixed-outline floorplanning: Enabling hierarchical design,” *IEEE TVLSI*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003.