Interconnect Estimation without Packing via ACG Floorplans*

Jia Wang and Hai Zhou

ECE, Northwestern University, Evanston, IL 60208, USA

Abstract—ACG (Adjacent Constraint Graph) is a general floorplan representation. The refinement of constraint graphs gives not only an efficient representation but also a representation sharing the advantage of adjacency graphs. As most edges in an ACG are between modules that are close to each other, the physical distance of two modules can be measured without packing by the shortest path between them on the ACG. Experimental results verified this relationship and possible approaches for interconnect planning are discussed.

I. INTRODUCTION

Many years of active researches on floorplanning have accumulated a lot of techniques in the literature [11]. With the rising new opportunities for interconnect planning, many new representations came out recent years, for example, sequence pair [9], BSG [10], O-tree [4], B*-tree [2], corner block list [5], TCG [8], twin binary sequences [13], etc..

Generally speaking, there are two different approaches to floorplan design. A *constructive method* constructs a floorplan directly from module connectivity and module sizes and an *iterative method* iteratively modifies a floorplan into another and looks for a better one during the process. With the success of early simulated annealing floorplanners, iterative approaches have become the main stream in floorplanning. Almost all floorplan representations published in recent years mainly focused on: 1. whether the representation is complete—that is, having all possible floorplan represented; 2. how small is the redundancy of the representation—measured by the number of representations producing the same or inferior solutions; and 3. how efficiently a floorplan is computed from a representation. Obviously, these three questions are the only concerns for iterative approaches.

On the other hand, the research in constructive approaches is less active. Since module connectivities are used for floorplan construction, the adjacency graph (also known as Grason graph [3]) plays a crucial role in these approaches [6, 1, 7]. In an adjacency graph, an edge represents the adjacency between modules that share a boundary. Therefore, if two modules have an edge in the adjacency graph, the interconnect between them will be short. However, to be an adjacency graph, a graph must satisfy many properties: being planar, triangulated, and without complex triangles. Starting from a connectivity graph, a constructive approach will usually first create a planar graph by deleting some edges or introducing some nodes, then triangulate it and delete complex triangles. Some of these steps are NP-complete [12]. Even though there exists a linear time algorithm in theory [1], generating a floorplan from an adjacency graph is also not an easy process.

ACG [14] bridges these two schools of floorplanning research. In iterative approaches, the generation of the floorplan from a representation usually involves a compaction (for example, from an O-tree, a module's position is found as smallest x and y satisfying the constraints). Therefore, area is implicitly considered first. Since ACG is a constraint graph, the floorplan area and the module positions can be efficiently evaluated by longest path computations. In constructive approaches, interconnects are first explored to generate an adjacency graph. They are more suitable for interconnect planning. But calculating module positions from an adjacency graph is more

complex. A central concept in module compaction is a constraint graph which is used to forbid overlaps among modules. By combining the adjacency graph and the constraint graph, ACG represents both interconnects and areas.

As shown in [14], ACG is superior for an iterative approach because it is complete, non-redundant, and efficient to map to a floorplan. However, we consider its main merit to be its simplicity and direct representation of geometry. Since most edges in an ACG only connect modules that are close to each other, which will be explained later in the paper, we identify those edges and construct a simplified graph called Reduced ACG to capture the adjacency in geometry. In a Reduced ACG, a shortest path between two modules is a good estimation of the distance between them, which is confirmed by our experiments.

II. ACG-ADJACENT CONSTRAINT GRAPH

In a constraint graph, vertices represent modules and an edge in horizontal graph represents “left to” relation and an edge in vertical graph represents “below” relation. It is far too common for one module to be both left to and below another module in a floorplan. In this case, both horizontal and vertical edges are presented. Adjacency graph, on the other hand, is an undirected graph and has one edge between each pair of adjacent modules. As an illustration, for a floorplan given in Figure 1(a), its constraint graph is shown in Figure 1(b) and its adjacency graph in Figure 1(c).

Fig. 1. (a) A floorplan; (b) Constraint graph; (c) Adjacency graph; (d) ACG-Adjacent Constraint Graph.

Obviously, the adjacency graph of a given floorplan is an undirected sub-graph of its constraint graph. However, a constraint graph over-specifies. In the constraint graph in Figure 1(b), both “below” and “left to” relations are specified between modules b and c , where there is no adjacency. It is obvious that over-specification has no benefit in a representation. The essential idea of constraint graph, that is, forbidding module overlap, can be maintained by requiring that there is one relation (“left to” or “below”) between any two modules.

Another set of redundant edges in a constraint graph are the transitive edges. For example, in Figure 1(b), the edge (a, e) is implied by edges (a, c) and (c, e) , thus is a transitive edge. Transitive edges are not necessary for floorplan construction. On the contrary, since they are between modules far away from each other, their presence gives noises to the geometrical proximity information in other edges. For example, after deleting edges (a, e) , (b, e) , and one (b, c) from Figure 1(b), we arrive at a graph shown in Figure 1(d), where solid edges represent horizontal relations and dashed edges vertical ones. Notice how close this graph is to the adjacency graph.

Definition 1 A cross is a subgraph on four nodes a, b, c, d such that $(a, b), (c, d)$ are one type of edges (e.g. vertical) but $(a, c), (b, c), (a, d), (b, d)$ are the other type (e.g. horizontal).

*This research was supported by NSF under CCR-0238484.

Definition 2 An ACG (Adjacent Constraint Graph) is a constraint graph that

1. has exact one relation (horizontal or vertical) between each pair of vertices;
2. has no transitive edge;
3. has no cross.

We group the edges in an ACG according to the relations they represent and call them groups H and V . From the definition, there are paths between any two modules within exact one group.

III. PROPERTIES OF ACG

A. Total order and symmetry

Theorem 1 The directed edges in an ACG form a total order on the vertices. In other words, the vertices can be arranged in an order such that all the edges go from left to right.

Total symmetry is another elegant property of ACG. Formally, it can be stated as the following proposition.

Proposition 2 Given an ACG data structure, it is still a valid ACG when the edge directions are reversed, or the edge groups H and V are exchanged.

This symmetry comes directly from the symmetry in the geometrical relations represented. The four geometrical relations—left, right, up, down—are symmetrical with each other. Based on the symmetry, every provable result concerning edges in an ACG also implies the other three dual results. It simplifies our presentation in the sequel.

Corollary 2.1 If a result concerning edges in an ACG is correct, then the result by exchanging left and right (and forward and backward) or H and V is also correct.

Based on the above theorem and proposition, we organize an ACG as follows. The vertices will be doubly linked in a linear order (the total order). Edges are all directed from left to right. Each edge keeps its two end vertices and is kept in one edge list at each end vertex. Each vertex maintains four linked lists of edges: one for incoming H edges, one for outgoing H edges, one for incoming V edges, and one for outgoing V edges. The edges in each list are ordered according to the distances between end vertices: shorter edges come first. The structure is illustrated by an example in Figure 2(a), where edges are shown in arcs: the edges above the vertex line are in group H and those below are in group V . Notice that vertex d has one incoming H edge from a , one outgoing H edge to e , two incoming V edges from b and c , and no outgoing V edge. These four lists of edges have direct geometrical meanings: each connects to constraining modules in one direction: left, right, top, and bottom. And the edge orders in the lists will be either clockwise or counter-clockwise, based on how H and V edges are interpreted. For example, if the H edges are interpreted as “left to” and the V edges as “below”, the geometrical interpretation of Figure 2(a) is illustrated in Figure 2(b), where H edges are ordered counter-clockwise and V edges are ordered clockwise.

Referring to the linked lists belong to a vertex a , we have the following lemma regarding the relationship between H edges and V edges.

Lemma 1 If (a,b) and (a,c) are two consecutive H edges starting from a , (b,c) must be a V edge.

Fig. 2. (a) ACG structure; (b) Geometrical relations.

Fig. 3. Patterns of a cross.

B. Cross elimination and edge classes

Based on Definition 1, the patterns of a cross in the data structure are shown in Figure 3. We notice here that any vertex has edges to the three other vertices. Furthermore, we can prove the following theorem.

Theorem 3 If the cross formed on a, b, c, d is minimal, that is, no other cross exists on vertices between a and d , the vertices b, c, d are consecutive among the neighbors of a , that is, except c , no vertex between b and d is connected to a .

Theorem 3 implies that if we follow the edges of a in consecutive order, an edge type pattern “VHH” or “HVV” results in a cross. So the edge type pattern should be either “VV...VHVH...” or “HH...HVHV...”. Thus we could distinguish all the ACG edges into four classes.

Definition 3 An edge (a,b) of ACG must belong to exactly one of following four classes, where we refer to the edge type pattern of the outgoing edges of a .

- 1 The first V (H) edge, i.e., a and b are consecutive vertices and (a,b) is belong to group V (H).
- 2 The following V (H) edges before the first H (V) edge.
- 3 The first H (V) edge.
- 4 The edges follows the first H (V) edge.

Every vertex except the last one has exactly one class 1 edge, zero or more class 2 edges, at most one class 3 edges, and zero or more class 4 edges.

Symmetry is available to those classes.

Theorem 4 If we reverse all the edges, edges belong to class 1 or 4 are still belong to class 1 or 4 respectively. Class 2 edges become class 3 and class 3 edges become class 2.

The following two theorems state the relationship between the end points of edges starting from a .

Theorem 5 For a vertex a , suppose (a,b) and (a,c) are two consecutive edges where (a,c) is a class 2 or 3 edge. Assume (a,b) is a V edge. Then (b,c) is a H edge.

Theorem 6 For a vertex a , suppose (a,b) , (a,c) , and (a,d) are three consecutive edges where (a,d) is a class 4 edge. Assume (a,b) is a V edge, (a,c) is a H edge, and (a,d) is a V edge. Then (b,c) and (b,d) are two consecutive H edges starting from b . And the edges (c,d) and (a,d) are two consecutive V edges end at d .

According to Corollary 2.1, Theorem 5 and Theorem 6 are also true when we change V to H and H to V .

Figure 4 shows an example for these two theorems along with the edge classes. With these two theorems, we could append a new vertex to an existing ACG and build a valid ACG in linear time and storage complexity in terms of the number of the edges added.

Fig. 4. An example for Theorem 5 and Theorem 6. The numbers indicate the classes of the edges starting from a .

IV. REDUCED ACG

According to Theorem 6, all the class 4 edges could be implied if the class 1,2,3 edges are presented. We obtain a simplified graph by removing all the class 4 edges.

Definition 4 A Reduced ACG is obtained by removing all the class 4 edges from an ACG.

Since a Reduced ACG is a subgraph of an ACG, a partial order must exist between the vertices. Further more, the class 1 edges ensure the existence of the total order. The total symmetry is also kept because of Theorem 4. We summarize these two facts in the following theorem.

Theorem 7 The directed edges in a Reduced ACG form a total order on the vertices. Given a Reduced ACG data structure, it is still a valid Reduced ACG when the edge directions are reversed or the edge groups H and V are exchanged.

Based on this theorem, we could reuse the ACG data structure for the Reduced ACG. It is possible to rebuild the corresponding ACG from the Reduced ACG by using Theorem 6, resulting in Theorem 8.

Theorem 8 There is a one-to-one mapping between ACG and Reduced ACG.

Like building an ACG, this procedure also takes linear time and storage in terms of the number of the edges added.

In addition to obtaining a Reduced ACG from an ACG, we could build a Reduced ACG by appending new vertices one by one using Theorem 5. In term of the number of the edges added, the time and storage complexity are both linear. Actually, according to Theorem 9, the total time and storage required to build a Reduced ACG is linear to the number of the vertices.

Theorem 9 The number of the edges in a Reduced ACG is at most $3n$, where n is the number of the vertices.

V. INTERCONNECT PLANNING WITH REDUCED ACG

When most edges in a Reduced ACG also represent the closeness of the modules, we can use them for interconnect planning. A constructive approach may start with a connectivity graph of the modules and select as many as possible edges into a Reduced ACG based on their weights of connections. An iterative approach needs to solve the problem of interconnect estimation on a given Reduced ACG. Interconnect estimation may include the wire lengths, path delays, and routing congestion.

Commonly, we use the physical information from the placement of the modules for the interconnect estimation. As we can generate an ACG which is a constraint graph from a Reduced ACG, the flexibility on module positions within the placement could be easily explored by the longest path computations to help interconnects. However, it is possible that interconnect estimation can be done directly on the Reduced ACG without generating an ACG and then performing the longest path computing.

The general idea for the interconnect estimation without packing is as follows. Taking two vertices a and b from a Reduced ACG, we are going to estimate the length of the wire connecting a and b . This wire length is computed as the Manhattan distance of the centers of the two modules when the positions of the modules are fixed under a packing procedure, which is always time consuming. Suppose a is in front of b according to the total order of the vertices. We treat the Reduced ACG as a directed graph where every edge goes from the left to the right. Then there is at least one path from a to b . An example of such a path is the path made up of the class 1 edges from a to b . Every edge along the path is either a “left to” or a “below” relationship. The length of the path, which is defined as the number of the edges in that path, can be used as an estimation of the physical wire length. When there are several such paths, we pick up the one with the minimal length, which means we always look at the shortest path between two modules. Actually, a breadth-first-search from a will be enough to obtain the shortest path from a to b . The physical path delay can be similarly estimated by adding all the wire length estimations along the path itself. We call this estimation *general estimation*.

The large dead space could result in bad estimations where white spaces contribute a large amount to the wire lengths. However, those floorplans with large dead space will be excluded when we are optimizing for area or interconnect. So we can limit our discussion on estimations for floorplans with moderate dead space. More specifically, we consider only the floorplans whose dead space is no more than the total area of the modules.

A more accurate but complex estimation can be done by including the sizes of modules when we calculate the length of a shortest path. Suppose a is in front of b according to the total order of the vertices. For every vertex c between a and b in the total order, we use dynamic programming to determine the length of the shortest path from a to c . The H (V) edge length used to compute the path length is equal to half of the sum of the widths (heights) of the two modules connected by this edge. We call this estimation *accurate estimation*.

Another possible estimation could be done by treating the Reduced ACG as an undirected graph instead of a directed one. Since more edges are considered, the lengths of shortest paths are smaller. However, there may be problems on the accuracy of the estimations. For example, suppose $t_1, t_2, \dots, t_6$ are arranged in a line from the left to the right and b is below all of them. When we treat the Reduced ACG as a directed graph, we could give good estimations to the lengths of the wires between each pair from $t_1, t_2, \dots, t_6$. However, when using the undirected version, b bridges every pair so that the estimations are not accurate any more. Therefore, we propose to use only the directed Reduced ACG for interconnect estimation.

VI. EXPERIMENTS FOR INTERCONNECT ESTIMATIONS

The following experiments have been done to verify the relationship between the estimations and the physical wire lengths. The computation of the Manhattan distances between two modules is performed after the positions of the modules are determined by packing to the bottom-left corner. Since what we have is a Reduced ACG, we need to rebuild the corresponding ACG and use this ACG for packing. For calculating the shortest path between two modules, we use both the general estimation and the accurate estimation.

We pick up three GSRC benchmarks (n100, n200, and n300). For every benchmark, we use simulated annealing for area optimizations like the one in [14] to obtain a floorplan whose dead space is no more than the total area of the modules. In those three floorplans, the dead spaces are almost the same as the total area of the modules, i.e., 50% dead space. We expect better results from smaller dead space. For every pair

of modules in each floorplan, we calculate the lengths of the shortest path between them under the two estimation methods as well as the Manhattan distance. The correlation between wire lengths and their estimations by the general estimation are shown in Figure 5, 6, and 7. The correlation between wire lengths and their estimations by the accurate estimation are shown in Figure 8, 9, and 10.

Fig. 5. General estimation for n100.

Fig. 6. General estimation for n200.

Fig. 7. General estimation for n300.

Those graphs exhibit good fidelity between the lengths of the shortest paths which are the estimations and the Manhattan distances which are the wire lengths under the dead space constraint.

REFERENCES

- [1] J. Bhasker and S. Sahni. A linear algorithm to find a rectangular dual of a planar triangulated graph. In *DAC*, pages 108–114, 1986.
- [2] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu. B*-trees: A new representation for non-slicing floorplans. In *DAC*, pages 458–463, 2000.
- [3] J. Grason. *A Dual Linear Graph Representation for Space-Filling Location Problems of the Floor-planning Type*. MIT Press, Cambridge, MA, 1970.
- [4] P.-N. Guo, C.-K. Cheng, and T. Yoshimura. An O-tree representation of non-slicing floorplan and its applications. In *DAC*, pages 268–273, 1999.

Fig. 8. Accurate estimation for n100.

Fig. 9. Accurate estimation for n200.

- [5] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, C.-K. Cheng, and J. Gu. Corner block list: An effective and efficient topological representation of non-slicing floorplan. In *ICCAD*, pages 8–12, 2000.
- [6] K. Kozminski and E. Kinnen. Rectangular dual of planar graphs. *Networks*, 15:145–157, 1985.
- [7] Y. T. Lai and S. M. Leinwand. Algorithms for floor-plan design via rectangular dualization. *IEEE TCAD*, 7(12):1278–1289, 1988.
- [8] J.-M. Lin and Y.-W. Chang. TCG: A transitive closure graph-based representation for non-slicing floorplans. In *DAC*, pages 764–769, 2001.
- [9] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. Rectangle-packing based module placement. In *ICCAD*, pages 472–479, 1995.
- [10] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani. Module placement on BSG-structure and IC layout applications. In *ICCAD*, pages 484–493, 1996.
- [11] R. H.J.M. Otten. What is floorplan? In *ISPD*, pages 201–206, 2000.
- [12] G. Yeap and M. Sarrafzadeh. Floorplanning by graph dualization: 2-concave rectilinear modules. Manuscripts, 1993.
- [13] 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*, 22(4):457–469, April 2003.
- [14] H. Zhou and J. Wang. Acg-adjacent constraint graph for general floorplans. In *ICCD*, 2004.

Fig. 10. Accurate estimation for n300.