AN EDGE-ORIENTED COMPACTION SCHEME BASED ON MULTIPLE STORAGE QUAD TREE

Pei-Yung Hsiao and Wu-Shiung Feng

Institute of Electrical Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.

ABSTRACT

In this paper we describe an efficient edge-oriented, error-tolerance and mixed-constraint graph compactor based on a simple and fast region-query data structure--Multiple Storage (M.S.) Quad Tree. Besides that the essential features of the M.S. quad tree data structure are that they support a dexterous region search and inherently can be used to handle the extended objects such as rhomboids and polygons, the constraint-graph compactor declared as the best flexible 1-D compaction model is improved in both fast building and accurately solving on the mixed constraint graph by the elegant assistances of the M. S. quad tree. Algorithms for generating the directed graph in adjacency list to be coordinated with the quad tree manipulations and the edge-oriented line-scanning operations are proposed in details.

INTRODUCTION

As the complexity of integrated circuit design increases, the need for a computer-aided layout editing tool becomes critical. However, the most critical role in a VLSI CAD layout tool is known as the compactor [1]. It is obvious that among the three typical 1-D compaction approaches including constraint graph, shear-line, and virtual-grid, the constraint-graph appears to be the most general and has turned out to be the most popular in recent years. The constraint-graph approach consists of two main processes, such as :

(1). Building the constraint-graph to indicate the relative locations and the design rules required among the elements, and
(2). Solving the constraint-graph to minimize the space by using the longest path method.

Generally, the specialized data structure, called corner stitching, has great difficulties with overlapping rectangles which severely limit its usefulness. But this problem does not exist in the M.S. quad tree as presented in this work. Moreover, the "corner stitching" inherently fails on the representation of polygons[2]. But with the multiple storage quad tree, it is just an easy work.

In addition to the extreme combination of the M.S. quad tree and the constraint-graph compaction, the error tolerance [3] and mixed constraint [4] functions are put into this system. Without caring any design rule or whether the layout being tiny, the only thing one should pay his attention to is the relative topology of the layout. The error tolerance function will correct the final layout and rearrange it automatically. If some specific constraints (not the design rules) are wanted in the layout, the mixed constraint function would solve them.

MULTIPLE STORAGE QUAD TREE

The primitive prototype of the quad tree got its name from the fact that it divides the plane into quadrants repeatedly until the quadrants are small enough that each one contains only a few objects. In 1986, an alternative method of storing the object intersected more than one quadrant was proposed by R. L. Brown [5-6]. Here, the objects intersecting more than one quad are stored respectively in their Qtreenodes of the quadrants. A quad tree of this type will be called a MULTIPLE STORAGE quad tree. It is obvious that the memory-space to be increased from multiple storage can be reduced to be negligible by storing pointer to the object rather than the object itself. The most important advantage of the multiple storage quad tree is that it brings a very efficient computing time of region query, $O(k/2 * \log N)$, for even small region. Where N is the total number of objects and k denotes those objects wholly or partially inside the specified window.

The multiple storage quad tree contains two kinds of nodes, treenode and object-reference node. A treenode, as shown in Fig.1, always consists of four pointers to point to subquads and four used-flags

CH2458-8/88/0000-2435$1.00 © 1988 IEEE

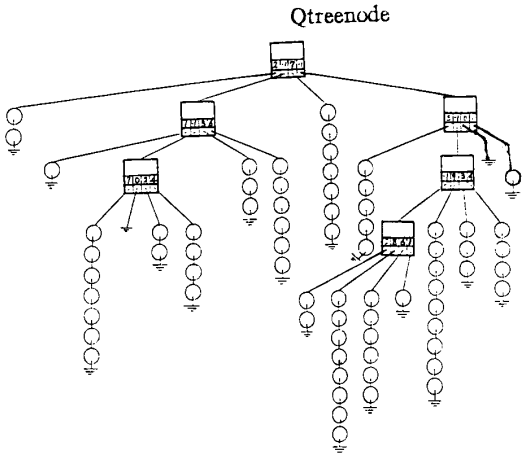

Fig.1. Quad tree node.

to represent the states of subquads. Since the number of treenodes are much smaller compared to the number of objects (N), the size of a treenode will become less important. The object reference node of the multiple storage quad tree consists of only two pointers. One pointer points to the substantial object and the other points to the next object reference node in the same leaf quad as illustrated in Fig.2.

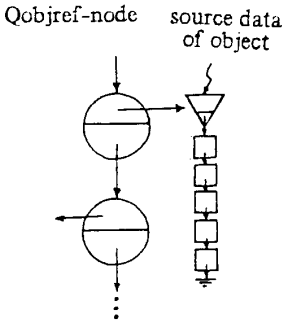

Fig.2. Object reference node.

Due to the limitation of the threshold value, the quad tree may split into subquadrants to keep away from overflowing. Fig.3 illustrates the numbe r of object changes within each quadrant before as well as a specified rectangle (object) is deleted from the layout plane individually. Meanwhile, it is easy to exhibit the corresponding representation the adaptive multiple storage quad tree.

Another special consideration in our approach is that by using several essen-

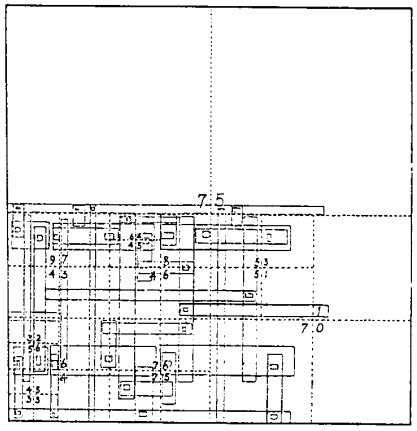

Fig. 3 The numbers stand for how many objects inside each subquadrant.

tial functions of M.S. quad tree, such as: qfind(), pseudo-qfind1() and pseudo-qfind2(), the event point schedules are dynamically updated during the execution of the plane-sweep algorithm. This approach brings our compactor to be free from pre-sorting the input objects before compaction. Many similar applications for the essential functions of M. S. quad tree can be broadly strewn in the field of CAD, computational geometry or image processing.

SOLVING ALGORITHMS ON COMPACTION

The algorithms used to preprocess the data structure of the graph_theoretic compactor, in other words, to construct the directed graph, were refined from the idea of S. L. Lin & J. Allen in M.I.T. [7], except the critical improvements from the introduction of the multiple storage quad tree. By the aids of the region queries of the multiple storage quad tree, the event schedule of the sweeping line is dynamically updated one by one and the algorithm may formulate the directed graph from a mask layout diagram smoothly.

Each time after all of the right and left edges encountered have been processed, the sweeping line automatically processes to the next step by the aids of the quad tree finding functions (e. g. qfind, sqfind1, sqfind2 and inclqfind) in a reasonable speed.

On considering the current sweeping line A as presented in Fig. 4, the first calculated step-value will support the sweeping line to jump at position C. However, with more accurate controlling of the quad tree finding-functions, the sweeping line may quickly jump back to the exact location, at position C, without tri-

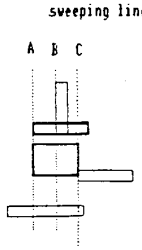

Fig. 4 The dynamic event point schedule of correction must be A to B but not A to C (qfind(), sqfindl() and sqfind())

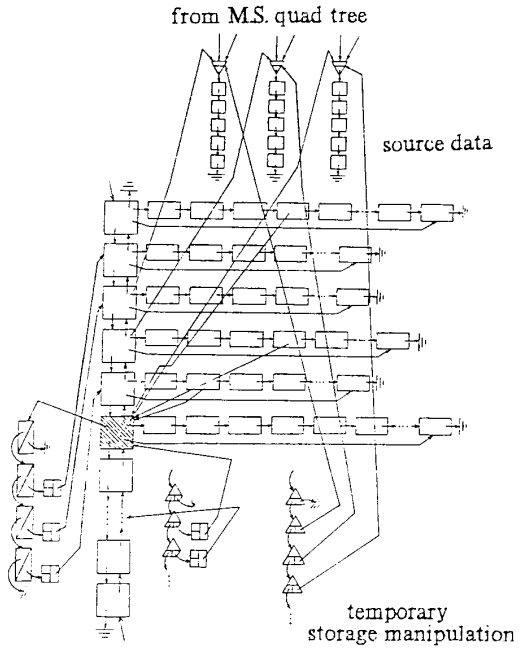

Fig. 5 Dynamic data structure for generating the adjacency list of the constraint graph.

vial considerations. Although the temporary states of the whole data structure during graph formulation are dynamic and very hard to be understood in nature, with the aids of the following appended data structure as shown in Fig. 5, it will become more easy to figure out the essential design scheme in this article.

The graphs obtained from the preprocessing as presented in this paper of Fig. 6 are the acyclic, directed, multi-end and nonplanar (one visibility) graph. The nonplanar graph is more indeed complex than the planar one. Almost all of the traditional compactors treat the constraint-graph as a planar graph (zero visibility graph) by simplifications during the operations of layout before compaction. Without the simplifications, the edge-oriented compactor (E.O.C.) still keeps a most excellent performance, as illustrated in Fig. 7.

CONCLUSION AND RESULTS

To make an intimate combination of M.S. quad tree and the E.O.C. is the most high unprecedented creativity and contribution

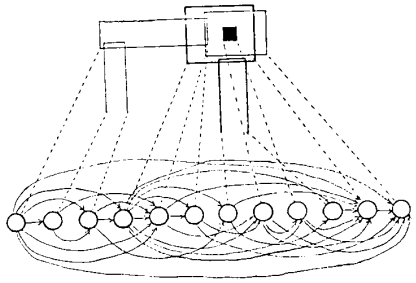

Fig. 6 Graph representation of the layout in the x-direction.

of this system. Without sustaining the fast region query data structure, M.S. quad tree, this algorithm of E.O.C. can not spend so less CPU running time to complete the compaction processing. With the help of the multiple storage quad tree, the operations of the layout editor system, such as object-move, object-rotate, object-copy, object-find, window-pan and window-zoom, become very time-saving.

The E.O.C. is implemented in C language and executed on a SUN III/110 workstation. In many instances, by swapping several technology-dependent subprograms, the same layout may be compacted with another distinct set of design rules. The choice of x-direction or y-direction compaction first may produce different final compacted layout. Clearly, it depends on the distinguished characteristics of the layouts.

In spite of the fact that many compactors treat the topology of layout by stick diagram, our system deals with the layout by the objects themselves without the simplification of stick diagram. This brings our constraint graph as an ONE-VISIBILITY GRAPH. Fig. 7 showst hat the CPU running time of E.O.C. is exactly close to linear, O(N), experimentally. However, the theoretical worst-case computing time required for E.O.C. is of O(N*log N) [8], where N is the number of individual objects in the layout. The worst-case comes from each encountered left edge possesses itself of an order of N connection constraints. Fortun-

ately, that is an impossible case in the real physical layout. And the following results reasonably shows our system as an linear compactor.

On account of this, a number of layouts have been done in this system. Some of them are shown in Fig. 8. The CPU time spent for compaction is approximately proportional to the number of the processing objects. From the experimental results as listed in Fig. 7, if the target layout has N objects, then the E.O.C. needs about (N* 65)/1000 seconds to compact it.

This system also maintains a feature of hierarchical amalgamation (function of cell smashing). It makes our system is able to break the boundaries of several layout cells and assembling them into a large one. In summary, the most seven essential features of the improvement of the multiple storage quad tree onto the E.O.C. of this system are:

1). High unprecedented creativity.
2). Sustain a most up-to-date fast region query operation, M.S. quad tree, for both E.O.C. and editor.
3). Good performance.
4). Mixed constraint.
5). Error tolerance.
6). A substantially hierarchical amalgamation.
7). Without grid restriction.

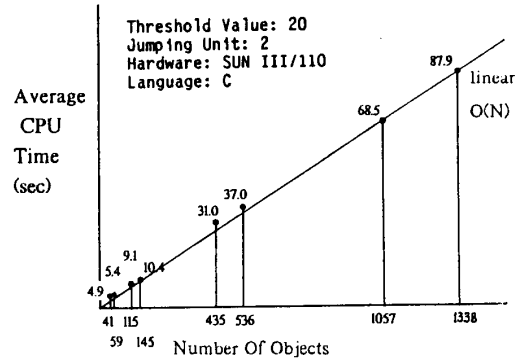

Fig. 7 Execution time of the E.O.C. for for different numbers of objects.

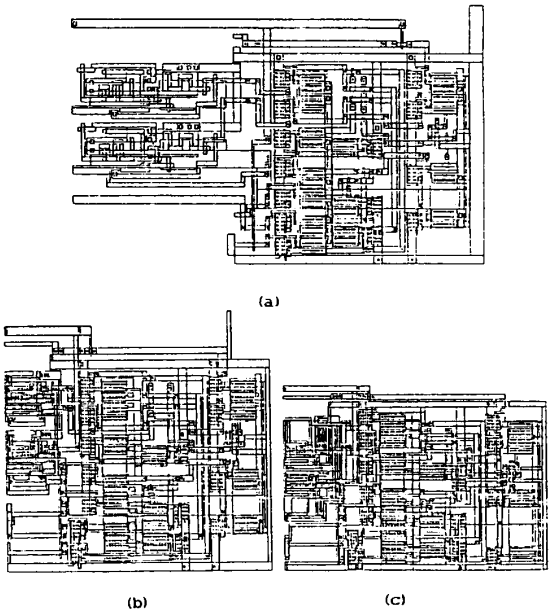

Fig. 8 CXTOMAG Adder (1057 objects) (a) source layout, (b) after X-compaction, and (c) after XY-compaction.

REFERENCE

[1] G. Kedem and H. Watanabe, "Graph-optimization techniques for IC layout and compaction, " IEEE Trans. Computer-aided Design, vol. CAD-3, pp. 68-86, Jan. 1984.

[2] J. K. Ousterhout, "Corner stitching: A data structuring technique for VLSI layout tools," IEEE Trans. Computer-aided Design, vol. CAD-3, pp. 87-100, Jan. 1984.

[3] C. Kingsley, "A hierarchical, error-tolerant compactor," in Proc. 21st Design Automation Conf., pp. 126-132, June 1984.

[4] Y. Z. Liao and C. K. Wong, "An algorithm to compact a VLSI symbolic layout with mixed constraints," in Proc. 20th Design Automation Conf., pp. 107-112, June 1983.

[5] J. B. Rosenberg, "Geographical data structures compared: A study of data structures supporting region queries", IEEE Trans. Computer-aided Design, vol. CAD-4, pp. 53-67, Jan. 1985.

[6] R. L. Brown, "Multiple storage quad trees: A simpler faster alternative to bisector list quad trees," IEEE Trans. Computer-aided Design, vol. CAD-3, pp. 413-419, July 1986.

[7] S. L. Lin and J. Allen, "Minplex -- A compactor that minimizes the Bounding rectangle and individual rectangles in a layout," in Proc. 23rd Design Automation Conf., pp. 123-130, June 1986.

[8] F. P. Preparata and M. T. Shamos, " Computational Geometry," Springer-Vrelag Inc., New York, USA, 1985.