INCREMENTAL CAD

Olivier Coudert
Monterey Design Systems
894 Ross Drive, Suite 100
Sunnyvale, CA 94089-1443
coudert@mondes.com

Jason Cong
Computer Science Dept.
University of California
Los Angeles, CA 90095
cong@cs.ucla.edu

Sharad Malik
Dept. of Electrical Engineering
Princeton University
Princeton, NJ 08544
sharad@ee.princeton.edu

Majid Sarrafzadeh
Dept. of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208
majid@ece.northwestern.edu

ABSTRACT

Comprehensive study of incremental algorithms and solutions in the context of CAD tool development is an open area of research with a great deal of potential. Incremental algorithms for synthesis and layout are needed when design undergoes local or incremental change. Often these local changes are made to react to local change in the design, correct local errors or to make local improvements in one or more of the design quality metrics. In this paper we outline fundamental problems in incremental logic synthesis and physical design. Preliminary solutions to a subset of these problems will be outlined.

1. INTRODUCTION

In present and next generation VLSI chips, geometries get smaller, clock frequencies increase, and on-chip interconnect gains increased importance. Fundamental issues such as wire-congestion and routability, crosstalk and coupling noise, transmission-line behavior, power consumption, reliability and yield, and their interaction are crucial factors in designing next generation (VLSI) CAD tools.

Traditionally, CAD tools have been used more or less independently at the system-level, high-level, logic-level, and layout-level, each with its own set of synthesis, verification, and analysis tool-set. Each tool unaware of the big picture and the mechanism linking them together. Complexity of present and next generation VLSI system, demand elimination of the iterative process forced by the tool independence and made it necessary to perform these tasks concurrently. To cope with the complexity of this merger, incremental algorithms are a must.

There is a scattered literature in the area of incremental and dynamic data structures and algorithms (e.g., see [10] for an ovreview). In the following section, we intend to briefly survey them with great emphasis in the area of physical design. We will then formulate various problems that are fundamental in incremental physical design. We also outline a set of open areas of research that have great potential for research and development.

2. FORMULATION AND SURVEY

To eliminate the potentially divergent design iterations EDA tools need to perform incremental design modification as well as incremental analysis and estimation of the results. Issues like back-tracking without having to recalculate (from scratch) where we have come from and some check-point management gain increased importance in such a scenario [49].

Incremental algorithms for synthesis and layout are needed when design undergoes local or incremental change. Often these local changes are made to react to local change in the design, correct local errors or to make local improvements in one or more of the design quality metrics. Mechanisms are needed to control the portions of the design that are exposed for optimization [2]. And algorithms are needed to obtain good incremental solutions in a short amount of time.

Many iterative algorithms such as simulated annealing, force directed algorithms, and maze routing can be easily modified to handle incremental floorplanning, placement, and routing. The main challenge is to decide which instances we need to apply these algorithms to and what are the quality/speed tradeoff. A small number of research results in the area of incremental layout have been reported in the recent past, focusing on floorplanning [12], placement [4], and FPGA routing [14, 43]. Incremental logic optimization has been studied in [2, 50], and incremental FPGA technology mapping in [8] .

A major difficulty in physical design is the cycling dependency and strong interaction between placement, synthesis, and routing. To estimate the timing, one needs to know the interconnect, which requires a placement, which itself should be timing aware. Simple logic optimizations (e.g., sizing and buffering) can improve the timing, but that may be done to the expense of area, which can create local congestion, which placement can take care of by spreading out the cells, which consequently may hurt the timing. Incremental capabilities are obviously not sufficient: one also need a consistent strategy so that placement, synthesis, and routing cooperate to meet the timing and congestion constraints.

As smaller geometry features are being used, the struggle for the interconnect to meet both the timing and congestion constraints is getting ever more difficult. Congestion is usually managed by placement, but new researchs are being done to use synthesis for congestion relief. The simplest form consists of using logic optimizations to produce some extra slack that can then be used by the router to reroute nets and relieve some overcongested area. The most aggressive forms consist of driving the resynthesis by a constrained interconnect topology.

In the next three sections, we focus on incremental logic synthesis, incremental placement, and incremental routing.

3. POST PHYSICAL DESIGN LOGIC SYNTHESIS

In this section we examine logic synthesis techniques that can be used after some physical design has been completed. Thus, this synthesis is actually a resynthesis that attempts

to use the physical informations to change the logic netlist so that it better meets the timing constraints imposed on it. A secondary goal may be to improve the congestion to help with routing. The amount of physical design completed may vary —at the least some rough placement information for the cells is needed. Typically the global route for the signals is also provided to help better estimate interconnect delay. The expectation is that the logic modifications will be small enough to not disturb this global route significantly. Detailed routing is generally not done yet —any logic change is likely to change the routing so much that it becomes useless. Given that the placement and possibly the global route is already done, the logic changes are expected to be fairly local, thus largely retaining the placement and global route. This limits the scope and nature of the resynthesis algorithms that can be used in this context. The actual amount of change permitted is a function of the specific flow in which they are used. However it is important to note that the amount of acceptable changes is much larger than what is expected in ECO (engineering change order synthesis), where only elementary changes are allowed to meet some changes in the specification [3]. Thus, while the body of work done in in-place optimization and ECO is related to the current context, it is also largely too limited.

We now examine specific techniques that have seen industrial success in this context.

3.1. Gate Sizing

The goal of gate sizing is to determine optimal sizes for the gates so that the circuit meets the delay constraints with the least area/power cost. A larger gate will have a higher drive strength (lower resistance) and hence will be able to charge/discharge output capacitances faster. However, it also has a higher input capacitance. This results in the preceeding gate seeing a larger capacitive load and thus suffering an increasing delay. Thus, sizing requires a careful balancing of these two conflicting effects, and the optimal solution will thus require the coordination of the correct sizes of all the gates along and off critical paths. This points to a global solution, where the sizes for all the gates in some critical section are determined simultaneously [36]. However, global techniques may be difficult to use directly in the post physical design context, since the changes in sizes as a result of resizing may result in a placement that cannot be made valid with small changes.

Analytical techniques exist for sizing in the continuous domain (e.g., linear programming [31], posynomial [15], convex programming [44]), and various attempts have been made to use these results with discrete sizes in a cell library based design style [18]. The theory of constant effort [48] and its more usable approximation, constant delay [16], provide a direct way to select cell sizes directly for each stage. For example, the optimal delay on a path is obtained by distributing the effort evenly among the logical stages. This means that if the delay of the whole path is fixed, then the delay of every stage must be kept constant. Thus, cell sizes can be selected to match this constraint by visiting all the gates in a topological order, starting from the outputs. However, intersecting or reconvergent paths often produce incompatible stage delays. Also, sizing at the physical level should use a realistic interconnect delay, and must consider the input slope effect, as well as the difference between falling and rising delays.

Alternatively, discrete sizing may be done using global search techniques [11]. These techniques attempt to reach a global optimal through a sequence of local moves, i.e., single-gate changes. While these are computationally expensive, they can take advantage of very accurate delay models, they can enforce complex constraints by rejecting moves that violate them (e.g., validity of the placement), and they can simultaneously combine sizing with other local logic optimizations (e.g., gate placement, buffering, pin swapping). They have been shown to provide good results for discrete libraries. They are particularly promising in the context of post-physical design synthesis, since they can focus on small critical sections and find a small sequence of moves that meets timing constraints while maintaining the validity of the placement.

Analytical sizing methods based on simple convex delay models are global and fast. Discrete sizing methods are slower, but very accurate and focused. Both methods should be used in the context of incremental synthesis, depending on the extend of the modification of the netlist, and on the level of accuracy of the physical informations.

3.2. Buffering

Buffering, like sizing, is really an electrical optimization and not a logical optimization in the sense that it does not change the logical structure of the netlist. Buffering serves multiple functions:

- Long wires result in signal attenuation. Buffers (also known as repeaters in this context) are used to restore signal levels.
- Like sizing, buffering can be used to increase the drive strength for a node that is driving a large load. A chain of one or more buffers can be used to drive a large load.
- Buffers can be used to isolate or shield signals on a critical path from high-load off-critical-path signals. In this case the buffer is used to drive the off-critical path load and the driver on the critical path sees a much reduced load of just the single buffer in addition to the critical signal

It is possible to come up with some ad-hoc solutions for each of the above cases. For example, repeaters can be added at fixed wirelength intervals determined by the technology. However, critical nets often need to be buffered to satisfy more than one of the above listed requirements. Thus, algorithmic solutions are very much desirable that balance the attenuation, drive strength and shielding requirements. This problem is hopelessly NP-hard. Even the problem of determining the best buffer tree for a given net ignoring the placement of the driver and the sink pins has been shown to be NP-hard [51]. An interesting solution exists for the case when the topology of the buffer tree is fixed and the potential buffer sites are fixed. This is the case when the global route for the net is already determined and the buffer tree must follow this route. In this case it been has shown that a polynomial time solution exists under the use of an Elmore delay model for interconnect [52] using a dynamic programming algorithm. Various attempts have been made to overcome the two limitations of this algorithm —the fact that the topology is already fixed, and that the Elmore delay model does not accurately capture the resistive effects of deep sub-micron technologies. The former is considered to be more of a problem, since fixing the topology can severely limit the shielding possibilities and lead to overall sub-optimal solutions. What is needed is the ability to determine the topology as part of the buffering solution. Some attempts have been made to develop heuristic solutions here (e.g., [38, 45]), including even adding additional

degrees of freedom like wire and driver sizing. While these techniques provide better solutions than the fixed topology approach, it is hard to say how close to the optimal these efforts are. This is a hard problem and continues to be a subject of active research. Various post-placement considerations have been included by researchers such as accounting for blockages in the routing topologies, and restricting the locations of buffers due to placement constraints (e.g., [57, 21]).

Determining the optimal buffer tree for a given net is only one part of the complete buffering problem. Given that buffering a given net can change the constraints (required time/slack, load) on the pins of another net, the final solution is sensitive to the order in which the nets are visited. In addition, once a net is buffered, the gates may no longer be optimally sized. Resizing gates before the next net is buffered can modify the buffering problem. Researchers have considered combining sizing and buffering into a single step [23], but again this problem is very complex and far from being considered as solved.

3.3. Technology Remapping

Technology mapping attempts to find the best selection of cells from a given cell library to meet a given delay constraint with the least area/power. Post-physical design mapping is a remapping step that attempts to find a better mapping by using the existing physical information for determining interconnect delay. The challenge here is to work on small sections so that the given placement is not significantly disturbed, yet at the same time be effective enough to improve the design. Mapping has been well studied [17], so the mapping algorithms are well known, knowing where to apply them is the key. One new aspect of the problem is determining where to place the new cells created during the remapping phase. Typically some simple solutions based on the fixed boundary locations are used during the mapping itself, with a clean up step to make the placement legal [42].

3.4. Logic Restructuring

Logic restructuring is the strongest technique in the suite of logic optimization techniques in the sense that it can significantly change the structure of the netlist. This also makes it the most difficult to apply in a post-physical design context where the changes are expected to be small to maintain the validity of the placement. However, the basic ideas in restructuring can still be used to improve the timing and congestion properties of the netlist as long as the changes are focused on key sections and the modifications are within the space of acceptable changes, e.g., the changes do not violate capacity/congestion constraints for various physical regions of the design. The restructuring techniques themselves range from well known applications of commutative, associative and distributive properties of logic functions [17], to sophisticated Boolean resynthesis (e.g., BDD-based). The challenge is in knowing where to apply them, and maintain the constraints imposed by the existing physical design. Several researchers have worked on this problem (e.g., [41]), however, with no dominant technique emerging.

4. PLACEMENT

The placement problem can be defined as follows. Given a circuit consisting of modules with predefined input and output terminals and interconnected in a predefined way, construct a layout indicating the positions of the modules so the estimated wire length and layout area are minimized. The inputs to the problem are the module description, consisting of the shapes, sizes, and terminal locations, and the netlist, describing the interconnections between the terminals of the modules. The output is a list of x- and y-coordinates for all modules. The main objectives of a placement algorithm are to minimize the total chip area and the total estimated wire length for all the nets. We need to optimize chip area usage in order to fit more functionality into a given chip area. We need to minimize wire length in order to reduce the capacitive delays associated with longer nets and speed up the operation of the chip.

Placement algorithms are typically divided into two major classes: constructive placement and iterative improvement. In constructive placement, a method is used to build up a placement from scratch; in iterative improvement, algorithms start with an initial placement and repeatedly modify it in search of a cost reduction. If a modification results in a reduction in cost, the modification is accepted; otherwise it is rejected. Constructive placement algorithms are generally very fast, but typically result in poor layouts. Since they take a negligible amount of computation time compared to iterative improvement algorithms, they are usually used to generate an initial placement for iterative improvement algorithms. One of the biggest challenge for placement tools are the fast growing circuit size. A good placement algorithm has to be more effective than ever to find a good layout as quickly as possible.

There are two classes of placement algorithms that has been studied in the area of concurrent placement and synthesis: pro-active placement makes the subsequent routing step easier. Cost function used in pro-active placement are, for example, wirelength, (congestion measures, global-routing measure. Re-active placement modifies a given placement based on the a given global or detailed routing or based on the moves made during in-place synthesis. In re-active placement, the goal is to create a good placement without modifying it too much. This is what we informally call incremental placement. Next, we give a formal definition of incremental placement [1]

Incremental Placement: Given an existing placement optimized with respect to a given metric (e.g., wirelength), modify the placement to improve it with respect to other metrics (e.g., congestion, timing, or detailed routability). Or, given an optimized placement and a set of changes to the netlist (e.g., due to technology remapping) modify the placement to improve it.

The notion of a mismatch between the incremental optimizer and the magnitude of the changes between successive instances is introduced in [24]. An explanation of why incremental optimizers sometimes work well and sometimes produce poor solutions is provided. However, better understanding of the problem change and algorithm capability are needed. For example, that fact that some incremental algorithms produce solutions that are better than the full-forced algorithms can be attributed to the strength of the algorithm, the nature of the problem itself, or can be due to the fact that a wider solution space being searched.

4.1. Incremental Placement in a Changed Netlist

Technology remapping, post technology retiming, gate adding/deleting are several possible reasons to change existing netlist. Suppose we already have a placement of an

[1] Traditional placement algorithms trying to optimize e.g., wirelength, congestion, timing at the coarse level can be classified as pro-active algorithms.

existing netlist. There are two methods to achieve a new solution on the changed netlist. The obvious approach is to simply run the placement tool again, while the alternative approach is to run an incremental placement algorithm which takes the previous solution as a starting point.

The following experiment is designed to study the behavior of the incremental placement in a changed netlist. Given a placement problem $P_0 = (C, N_0)$, and a solution S_0 produced by a wirelength driven placement tool, we modify the original placement problem P_0 to $P_1 = (C, N_1)$ by changing the netlist N_0 to N_1. Then a fresh placement run is performed on the new placement problem P_1. On the other side, an incremental placement algorithm is also applied on problem P_1, starting from the solution S_0. The results of these two placement runs are compared using the bounding box wirelength metric. For placement tool we use one of the best university tools, Dragon2000 [33]. For incremental placement we use greedy move-based local improvement approach [33].

To change the netlist, we randomly choose n pairs of *neighbornets* (two nets are *neighbornets* if and only if they are connected to a common cell). For each pair of nets we randomly select one cell from each net, exchange these two cells, and renew the connectivities of the cell pair. After changing the netlist, the total number of cells and the number of nets remain unchanged. We set n equal to 1%, 5% and 10% of the number of nets. Table 1 shows the experimental results on five MCNC benchmarks. Total bounding box wirelengths produced by two approaches are reported. As can be seen incremental approach works better for smaller changes on netlist while a full placement run is better if the netlist is changed significantly. However, note that the the incremental placement algorithm used here is a simple one and results might be different with more powerful incremental placers (yet to be developed).

Ckt	#nets changed					
	1%		5%		10%	
	full	*inc*	*full*	*inc*	*full*	*inc*
prim2	**3.54**	3.68	**3.55**	3.63	**3.55**	3.69
ind2	**14.3**	14.5	14.1	**13.4**	13.7	**12.9**
ind3	**41.3**	44.1	40.0	**39.5**	39.8	**38.8**
avqs	5.39	**5.33**	5.96	**5.89**	6.54	**6.01**
avql	**5.78**	5.79	6.25	**6.14**	6.77	**6.18**

Table 1. Total bounding box wirelength of placements produced by a full placement run (*full*) and an incremental run (*inc*). The better results are in boldface. Running time of the incremental approach is a small fraction of the full placement run.

4.2. Incremental Placement to Improve Timing

Table 2 shows a simple timing related experiment on MCNC benchmark circuits. A post processing algorithm takes a good placement produced by a wirelength optimization engine as the input, finds a subset of length nets (in terms of bounding box), reduces it to a given threshold value, e.g., to 90% of its original length. If there exist other nets with bouding box length larger than this threshold value, they will be also reduced. The algorithm we have implemented is greedy because it only accepts cell moves which do not increase the length of long nets. The changes of total wirelength are reported by the percentage increment on the original wirelength. Running time of the algorithm is negligible compared with a fresh placement run. The results show that it is not difficult to reduce the longest net by up to 20% without considerably increasing total wirelength. However, it gets very difficult to increase the wirelength beyond that. It should be noted that the implemented algorithm is greedy and very simple in nature. More effective algorithms are needed to tackle this problem.

circuit	**#cells**	**bouding box length reduced**			
		5%		**10%**	
		#nets	ΔWL	*#nets*	ΔWL
Primary2	2907	1	0.06%	3	0.37%
industry2	12142	2	0.06%	6	0.48%
industry3	15059	1	0.01%	1	0.04%
avqs	21854	1	0.03%	3	0.16%
avql	25114	1	0.06%	2	0.14%

circuit	**#cells**	**bouding box length reduced**			
		20%		**30%**	
		#nets	ΔWL	*#nets*	ΔWL
Primary2	2907	7	2.48%	13	N/A
industry2	12142	17	N/A	39	N/A
industry3	15059	1	0.06%	5	N/A
avqs	21854	5	0.78%	6	1.67%
avql	25114	10	1.85%	13	N/A

Table 2. Number of long nets and total wirelength increase in reducing bouding boxes length of long nets in a good placement, N/A means this can not be done by this greedy approach. Initial placement are produced by TimberWolf(without timing constraints)

4.3. Incremental Placement to Improve Congestion

Comparing with traditional placement objectives (net-cut, wirelength, etc.) congestion is least understood. However, it models routability more accurately than other objectives. Wang et. al. [53] proposed a consistent routing model defined by demand/supply relationship. Experiments show that the congestion objective is very ill behaved cost function such that directly using it will not produce low congestion placement. Since congestion and wirelength are globally consistent, a good way to reduce congestion is obtained by using a post processing stage after the traditional wirelength minimization stage. The post processing approach works more efficiently when the quality of input placement obtained in the previous stage is good. Minimizing congestion while maintaining the previous wirelength quality is very difficult. To minimize the wirelength changes caused by congestion reduction, a multi-center congestion minimization method is proposed [32]. By limiting the range of the congestion reduction, the original solution is changed locally so that the old objective is maintained. These congestion reduction techniques are incremental in nature. Again, optimizing one cost function while maintaining (or slightly degrading) other cost functions is a key point that needs further research.

5. ROUTING

Given the floorplanning and placement results, the routing problem is find out an exact implementation of all nets using conductive wires so that all the pins in each net are electrically connected. The connection wires have certain width and wire-to-wire clearance constraints, called *design rules*, determined by both processing technologies and performance optimization methods such as wire sizing and wire

spacing [9, 7]. In general, the routing problem are handled in a two-level hierarchy: *global routing* and *detailed routing*. In global routing, the entire routing region is partitioned into tiles or channels and a rough route for each net is determined in terms of the tiles or channels that the route passes through. The objective of global routing is typically to minimize the routing congestion and the total wirelength. The global routing results are used to guide detailed routing, which computes the final routing implementation of each tile or channel by determining the exact layer assignment, location, and dimension of every wire segment in that tile or channel for making all the connections.

Given an existing routing solution and a set of nets to be added or rerouted, the incremental routing problem is to find a new layout to accommodate these incremental routing changes with the minimal modification of the original routing solution. Naturally, the incremental routing problem can be divided into the incremental global routing problem and the incremental detailed routing problem. Incremental global routing is often used together with incremental logic synthesis, incremental floorplanning, or incremental placement to make sure that these incremental changes will not cause serious routability problems. Incremental detailed routing, also called ECO (engineer order change), may occur in several scenarios. For example, the netlist might be incrementally updated by the designer or synthesis tools after detailed routing. Or, some nets in a detailed routing solution might be determined to have timing/noise violation after detailed parasitic extraction and timing analysis, and need to be re-designed using different width, spacing, and/or buffering. In both cases, obviously, re-routing of all the nets is *too* time-consuming. Moreover, an entirely different layout may completely invalidate the existing extraction and detailed timing analysis results. Thus, the key problem in incremental routing is to preserve as much previous routing results as possible, while accommodating the new routing requests.

Both the incremental global routing problem and the incremental detailed routing problem can be partitioned into three related sub-problems.

- **Single Net Routing**. The first goal of incremental routing is to route the new nets without removing any of existing routed nets. This requires us to determine quickly for a given net, if it can be routed in the existing layout.
- **Rip-up and Reroute**. If the net can not be routed with existing nets, a rip-up and reroute operation will be carried out to free out more routing space so that all nets can be routed. The challenge of the rip-up and reroute problem is how to complete all the nets with minimal changes of the exiting routes (without changing the floorplan and placement results).
- **Incremental Floorplan and Placement Update**. If rip-up and reroute fails to complete all nets, the floorplan and placement of the design need to be updated to add more routing resources. The problem of incremental floorplanning and placement update is to minimize and localize the floorplan/placement changes while allowing all nets to be routed.

The solutions to these three sub-problems form a natural three-stage bottom-up flow for the incremental routing problem. The flexibility in each stage increases while the problem complexity also increases. In the following subsections, we shall first briefly review related results and then highlight one or two most promising solutions to each of the three problems. Open problems are also presented to motivate future research on this topic. Since the detailed design rules do not need to be enforced during incremental global routing, it can be considered as an "easier" problem than incremental detailed routing. So, the emphasis in the following subsections is on incremental detailed routing. Most techniques for incremental detailed routing, however, can be applied to incremental global routing, with proper simplification in many cases.

5.1. Single Net Routing

The single net routing problem (SNRP) is the easiest yet most fundamental one among the three problems. There are many methods to check whether a net is routable or not in a given layout. This is usually accomplished by actually *finding* a connection using a path searching algorithm in an abstract routing graph. This task is the kernel problem for *all* routing problems. The incremental routing problem is difficult in two aspects: First, the routing region is usually heavily congested with the existing routes; Second, the new route may not localize in a small routing region, which makes most path searching algorithms used by typical detailing routing system (work well for a tile or channel) inefficient or incapable to handle such a problem.

For single net routing, the routing region is normally reduced to a connection graph and the routing problem is mapped to a path searching problem in the graph. There are two ways to map the routing region to a graph: a tile-based approach and a point-based approach. In the tile-based approach, the routing area is partitioned into regions, called *tiles*, where the center-line of a path can pass through [46, 34, 30]. These tiles are defined by the boundary of the obstacles and stored using a corner-stitching data structure [39]. In the point-based approach, the routing area is populated with points where the center-line of a path can pass through [55, 37, 56]. A maze searching algorithm [28, 19, 47] is used to search a path on these graphs. In general, tiles are more complex to manage: tile-to-tile path needs post-processing to obtain a final design-rule correct route and there are some difficulties in using the tile-based algorithm for multi-layer routing with more complex design rules (see discussions in [6]). Thus, we turn our attention to point-based connection graphs for routing.

In early point-based routing algorithms, a uniform grid graph is used as the underlying routing graph. (This is usually also the case for global routing.) An explicit representation of the graph is used, that is, the graph is precomputed and stored. This approach is inefficient in current high-performance designs because the variable width and variable spacing design rules impose very fine grids on a uniform grid graph approach. Moreover, it is very inefficient to compute the routing graph for the entire layout while maybe only a small portion of it will be affected by incremental routing. Two approaches are proposed to improve the explicit graph: One is to find a minimal graph that guarantees to contain at least one shortest path if any such path exists [55, 54]. The drawback of the minimal graph approach is that it requires very expensive preconstruction. The other approach is to use compressed or implicit representation of the graph. A compressed representation of uniform grid-graph using segments is presented in [20]. Zheng, etc. al., presented an *implicit* representation of the routing graph, that is, their graph nodes are computed on-the-fly [56]. The underlying graph in their approach is an extension to the *track graph* introduced in

Ex.	Uniform	Non-Uniform Grid		Iroute [1, 40]	
	Expl. (MB)	Runtime (sec.)	Impl. (MB)	Runtime (sec.)	Mem (MB)
eco-1	160.2	19.1	10.9	42.15	32.7
eco-2	160.2	6.3	10.9	26.58	32.7
eco-3	160.2	34.5	7.2	68.70	32.6
eco-4	161.7	24.0	10.9	57.39	32.6
eco-5	191.0	12.3	12.7	43.14	35.2
eco-6	359.4	24.7	15.9	74.29	52.6
eco-7	641.1	38.2	43.6	79.79	84.7

Table 3. ECO test results: experimental results comparing implicit representation of non-uniform graph with explicit uniform grid graph and Iroute using seven ECO examples.

[55]. Their implicit approach, although very efficient in representation, is costly in computing the graph nodes.

A promising approach to the single net routing problem is reported recently uses a non-uniform grid graph (NUGG) with its implicit representation [5]. The graph is an orthogonal grid graph constructed based on the expansion of rectangular obstacles in the routing region according to wire/via width and spacing rules. The non-uniform grid graph, comparing to the one used in the uniform grid approach, is much smaller. What is more, the grid nature of such a graph makes it very easy to come up with an implicit representations — instead of pre-compute and store the graph, the graph is represented by two sorted array of its X and Y grid positions. To efficiently answer maze related queries, a two-level data structure using a first level "slit-tree" [25, 27] plus a second level interval tree [13] is applied. The query data structure is further enhanced with a *cache* structure that exploits the locality of the maze expansion.

The effective of this approach is validated in [5] where this graph and the auxiliary data structures is applied to the ECO problem. The paper compares the implicit graph and the query data structure with explicit uniform grid approach and Iroute, a well-known tile-based router for gridless routing used in the Magic layout system [1, 40], as shown in Table 3. The results show that not only this graph representation is very efficient in memory usage — $14\times$ smaller than explicit representation and $2-3\times$ smaller than Iroute. The queries into the data structure is also very fast. The run time of our maze routing algorithm is $2-4\times$ faster than Iroute.

5.2. Rip-up and Reroute Problem

When some nets can not be routed, a rip-up and re-route procedure is required to free out more routing resources and re-do the routing for the newly added nets and the nets that have been ripped up. Many algorithms have been proposed for rip-up and reroute [29, 26, 35]. However, most of them assume that there exists a underlying uniform routing grid and all net segments can be simplified as a zero width lines centered on the grid. This assumption makes it easy to model the resources in the routing region and simplifies the operation to exchange the resources between nets. It works well for global routing or grided detailed routing. However, it does not hold anymore in variable width and variable spacing routing. An accurate model of available routing resource in each local region and the flexibility to pick the re-routes globally are both needed to re-route in a gridless environment.

Example	Routed Nets			Run Time		
	n.b.n	w.p.	total	n.b.n.	w.p.	×
Block	489	496	496	4500.6	270.0	16.7
Mcc1	3939	3998	4004	9499.6	1365.1	7.0
Mcc1c	3931	3978	4004	5621.0	1508.5	3.7
Ray	409	418	430	518.8	172.0	3.0

Table 4. Routing results with wire planning: four examples are used to compare the completion ratio and run time for two approaches — net-by-net (shown as "n.b.n") and wire planning (shown as "w.p.").

Existing rip-up and re-route algorithms can be broadly categorized into the following two types:

- *Those always maintain design rule correctness*, such as the Pathfinder [35] and Silk [29];
- *Those allows temporary design-rule violation*, such as the "cross-and-touch" router [26]

There are some limitations if we strictly enforce design rule correctness in every step during routing. First, the result will rely heavily on the ordering of nets, as previously routed nets become obstacles for later ones. The rip-up and reroute algorithm has to be *smart*, or at least *fair* in selecting proper net orders. However, there is no obvious solutions other than simple heuristics and trial-and-error methods. Second, selecting nets to be ripped up is difficult, especially in incremental routing when the original routing algorithm used to generate the layout may not be available. The second type of rip-up and reroute algorithm is more flexible since by allowing design-rule incorrect routes, one can at least attempt to route all the nets and obtain a *global* picture of where the congested areas and free spaces are. Thus, many practical routing systems use this approach.

The key problem in rip-up and reroute for incremental routing is to find the solution with minimal changes in the previous nets. In a gridless routing environment, this problem is difficult in that the routing resources and previous routed wire can not be simplified as grids or tracks. We feel that the best solution to this problem is to develop a global control structure for the overall routing system that enables both the high-level planning prior to detailed routing and the rip-up and reroute after detailed routing. The congestion-driven *wire planning* algorithm presented in [22] provides the capabilities to take exact gridless design rules into consideration, model the available routing resources accurately, plan for the incremental routes prior to single net routing, and also re-plan in rip-up and reroute. Such a framework gives the rip-up and re-route algorithms global flexibility to evaluate the alternatives yet with detailed knowledge about the available routing resources in each region. The results, as shown in Table 4, show that the detailed routing system, compared to a net-by-net approach using the gridless detailed router developed in [5], is 3 – 17 times faster while the completion rate is also improved. These improvements are critical for applying the gridless detailed routing system in current and future VLSI designs where a true variable width and variable spacing router is needed.

5.3. Incremental Floorplan and Placement Update

When rip-up and reroute fail to route all the nets, we need to consider modifying the given placement and/or floorplan configurations. This results incremental floorplanning

and/or placement. The rip-up and re-route engine may suggest several possible regions for increasing routing resources in order to complete routing. The incremental floorplan and placement algorithms need to choose the right combination of routing regions for increasing the routing resources yet minimize the modification of the current floorplan/placement solution. Incremental floorplanning and placement update is a vital part of a incremental routing system because it provides the link between routing and floorplanning/placement. However, little progress has been reported for this problem, and we rank it as a high priority problem that needs more studies.

6. CONCLUSION

In this paper we discussed the need for incremental CAD algorithms (and effective software) when design undergoes local or incremental change. Often these local changes are made to react to local change in the design, correct local errors or to make local improvements in one or more of the design quality metrics. We outlined a set of problems in synthesis, placement, and routing and suggested possible solutions. Focused participation in research and development in the area of incremental and dynamic CAD is greatly needed and would help us cope with the complexity of present day VLSI systems and would faciliate concurrent optimization.

7. ACKNOWLEDGEMENT

The authors wish to thank graduate students at Northwestern and UCLA who have been involved in this project: Xioajian Yang, Maogang Wang, Jie Fang and Kei-Yong Khoo.

This works has been supported in part by grants from NSF MIP95-27389 and MIP-9357582 a grant from Fujitsu Laboratories at America under the California MICRO Program.

REFERENCES

[1] M.H. Arnold and W.S. Scott. An interactive maze router with hints. In *Proc. 25th Design Automation Conference*, pages 672–676, Jun 1988.

[2] D. Brand, A. Drumm, S. Mundu, and P. Narain. "Incremental Synthesis". In *Proceedings of the International Conference on Computer-Aided Design*, pages 14–18. IEEE, November 1994.

[3] D. Brand, Anthony Drumm, Sandip Kundu, and Prakash Narain. Incremental synthesis. In *IEEE International Conference on Computer-Aided Design*, 1994.

[4] C.-S. Choy, T.-S. Cheung, and K.-K. Wong. "Incremental Layout Placement Modification Algorithms". *IEEE Transactions on Computer Aided Design*, 15(4):437–445, April 1996.

[5] J. Cong, J. Fang, and K.Y. Khoo. An implicit connection graph maze routing algorithm for ECO routing. In *Proc. ACM/IEEE International Conference on Computer Aided Design*, pages 163–167, Nov 1999.

[6] J. Cong, J. Fang, and K.Y. Khoo. Via design rule consideration in multi-layer maze routing algorithms. In *Proc. International Symposium on Physical Design*, pages 214–220, Apr 1999.

[7] J. Cong and L. He. Theory and algorithm of local-refinement-based optimization with application to device and interconnect sizing. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 406–420, 1999.

[8] J. Cong and H. Huang. Technology mapping for field programmable gate arrays with incremental changes. In *IEEE-ACM Design Automation Conference*, pages 290–293, 2000.

[9] J. Cong, C.-K. Koh, and P. Madden. Performance optimization of VLSI interconnect layout. *Intergration, the VLSI Journal*, 21(1-2):1–94, 1996.

[10] J. Cong and Majid Sarrafzadeh. "Incremental Physical Design". In *International Symposium on Physical Design*, pages 84–92, 2000.

[11] Olivier Coudert. Gate sizing for constrained delay/power/area optimization. pages 465–472, December 1997.

[12] J. Crenshaw, M. Sarrafzadeh, P. Banerjee, and P. Prabhakaran. "An Incremental Floorplanner". In *Great Lakes Symposium on VLSI*, March 1999.

[13] H. Edelsbrunner. A new approach to rectangle intersections. *Intl. Journal of Computer Mathematics*, 13(3-4):209–229, 1983.

[14] J.M. Emmert and D. Bhatia. "Incremental Routing in FPGAs". In *IEEE International ASIC Conference and Exhibit*, 1998.

[15] J. P. Fishburn and A. E. Dunlop. TILOS: A posynomial programming approach to transistor sizing. In *IEEE International Conference on Computer-Aided Design*, pages 326–328, 1985.

[16] J. Grodstein, E. Lehman, H. Harkness, B. Grundmann, and Y. Watanabe. A delay model for logic synthesis of continuously-sized networks. In *IEEE International Conference on Computer-Aided Design*, 1995.

[17] Gary D. Hachtel and Fabio Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996.

[18] R. Haddad, L. P. P. P. van Ginneken, and N. Shenoy. Discrete drive selection for continuous sizing. In *IEEE International Conference on Computer Design*, 1997.

[19] F.O. Hadlock. A shortest path algorithm for grid graphs. *Networks*, 7(4):323–334, 1977.

[20] A Hetzel. A sequential detailed router for huge grid graphs. In *Proc. Design Automation and Test in Europe*, pages 332–338, Feb 1998.

[21] J. Hu and S. S. Sapatnekar. Simultaneous buffer insertion and non-Hanan optimization for VLSI interconnect under a higher order AWE model. In *International Symposium on Physical Design*, 1999.

[22] J. Fang J. Cong and K.Y. Khoo. DUNE: A multi-layer gridless routing system with wire planning. In *Proc. International Symposium on Physical Design*, 2000.

[23] Y. Jiang, S. S. Sapatnekar, C. Bamji, and J. Kim. Interleaving buffer insertion and transistor sizing into a single optimization. December 1998.

[24] A.B. Kahng and S. Mantik. "Mismatches of Incremental Optimizers and Instance Perturbations in Current Place-and-Route Tools". In *Proceedings of International Conference on Computer-Aided Design*, 2000.

[25] I. Kato, S. Ohhira, and Y. Hisatomi. A method of pattern data management of PWB layout system. In *Proc. 35th Annual Convention IPS Japan*, pages 2429–2430, 1987.

[26] K. Kawamura, T. Shindo, T. Shibuya, H. Miwatari, and Y. Ohki. Touch and cross router. In *Proc. of IEEE Conference on Computer-Aided Design*, pages 56–59, Nov 1990.

[27] E.S. Kuh and T. Ohtsuki. Recent advances in VLSI layout. *Proc. of the IEEE*, 78(2):237–263, Feb 1990.

[28] C.Y. Lee. An algorithm for path connections and its applications. *IRE Trans Electronic Computers*, EC-10:346–365, 1961.

[29] Y.-L. Lin, Y.-C. Hsu, and F.-S. Tsai. Silk: a simulated evolution router. *IEEE Transactions on Computer-Aided Design*, 8(10), Oct 1989.

[30] L.-C. Liu, H.-P. Tseng, and C. Sechen. Chip-level area routing. In *Proc. of Interational Symposium on Physical Design*, pages 197–204, Apr 1998.

[31] J. Jess M. Berkelaar. Gate sizing in mos digital circuits with linear programming. pages 217–221, 1990.

[32] K. Eguro M. Wang, X. Yang and M. Sarrafzadeh. "Multi-center Congestion Estimation and Minimization During Placement". In *International Symposium on Physical Design*, 2000.

[33] X. Yang M. Wang and M. Sarrafzadeh. "DRAGON2000: A Fast Standard-Cell Placement Tool". In *Proceedings of International Conference on Computer-Aided Design*, 2000.

[34] A. Margarino, A. Romano, A. De Gloria, F. Curatelli, and P. Antognetti. A tile-expansion router. *IEEE Trans. Computer-Aided Design*, CAD-6(4):507–517, Jul 1987.

[35] L. McMurchie and C. Ebeling. Pathfinder: a negotiation-based performance-driven router for FPGAs. In *Proc. of ACM Symposium on Field-Programmable Gate Array*, pages 111–117, Feb 1995.

[36] S. Manne O. Coudert, R. Haddad. New algorithms for gate sizing: A comparative study. In *IEEE-ACM Design Automation Conference*, pages 197–202, 1996.

[37] T. Ohtsuki. Gridless routers — new wire routing algorithms based on computational geometry. In *Proc. International Conference of Circuits and Systems*, 1985.

[38] T. Okamoto and J. Cong. Interconnect layout optimization by simultaneous steiner tree construction and buffer insertion. In *International Symposium on Physical Design*, 1996.

[39] J.K. Ousterhout. Corner stitching: a data-structuring technique for VLSI layout tools. *IEEE Trans. Computer-Aided Design*, CAD-3(1):87–100, Jan 1984.

[40] J.K. Ousterhout, G.T. Hamachi, R.N. Mayo, W.S. Scott, and G.S. Taylor. Magic: A VLSI layout system. In *Proc. 21st Design Automaton Conference*, pages 152–159, Jun 1984.

[41] M. Pedram and N. Bhat. Layout driven logic restructuring and decomposition. In *IEEE International Conference on Computer-Aided Design*, 1991.

[42] M. Pedram and N. Bhat. Layout driven technology mapping. In *IEEE-ACM Design Automation Conference*, 1991.

[43] S. Raman, C.L. Liu, and L.G. Jones. "A Timing Constrained Incremental Routing Algorithm for Symmetrical FPGAs". In *European Design and Test Conference*, 1996.

[44] P. M. Vaidya S. S. Sapatnekar, V. B. Rao and S. M. Kang. An exact solution to the transistor sizing problem for cmos circuits using convex optimization. pages 1621–1634, December 1993.

[45] A. Salek, J. Lou, and M. Pedram. A simultaneous routing tree construction and fanout optimization algorithm. In *IEEE International Conference on Computer-Aided Design*, 1998.

[46] M. Sato, J. Sakanaka, and T. Ohtsuki. A fast line-search method based on a tile plane. In *IEEE International Symposium on Circuits and Systems*, pages 588–591, May 1987.

[47] J. Soukup. Fast maze router. In *Proc. 15th Design Automation Conference*, pages 100–102, 1978.

[48] R. F. Sproull and I. E. Sutherland. Logical effort: Designing for speed on the back of an envelope. In *Proceedings of the IEEE Advanced Research in VLSI Conference*, 1991.

[49] L. Stok, D.S. Kung, D. Brand, A.D. Drumm, A.J. Sullivan, L.N. Reddy, N. Hieter, D.J. Geiger, H. Chao, and P.J. Osler. "BooleDozer: Logic Synthesis for ASICs". *IBM Journal of Research and Development*, 40(4):407–430, July 1996.

[50] G. Swamy, S. Rajamani, C. Lennard, and R.K. Brayton. "Minimal Logic Resynthesis for Engineering Change". In *International Symposium on Physical Design*, pages i1596–1599. IEEE, 1997.

[51] H. J. Touati. *Performance Oriented Technology Mapping*. PhD thesis, University of California, Berkeley, 1990.

[52] Lukas P. P. P. van Ginneken. Buffer placement in distributed RC-tree networks for minimal Elmore delay. In *IEEE International Symposium on Circuits and Systems*, 1990.

[53] M. Wang and M. Sarrafzadeh. "Behavior of Congestion Minimization During Placement". In *International Symposium on Physical Design*, pages 145–150. ACM, April 1999.

[54] P. Widmayer. On graphs preserving rectilinear shortest paths in the presence of obstacles. *Annals of Operations Research*, 33(1-4):557–75, Dec 1991.

[55] Y.F. Wu, P. Widmayer, M.D.F Schlag, and C.K. Wong. Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles. *IEEE Trans. Computers*, C-36(3):321–331, Mar 1987.

[56] S.Q. Zheng, Joon Shink Lim, and S.S. Iyengar. Finding obstacle-avoiding shortest paths using implicit connection graphs. *IEEE Trans. Computer-Aided Design*, 15(1):103–110, Jan 1996.

[57] Hai Zhou, Martin Wong, I-Min Liu, and Adnan Aziz. Simultaneous routing and buffer insertion with restrictions on buffer locations. In *IEEE-ACM Design Automation Conference*, 1999.