Incremental Physical Design

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

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

ABSTRACT

Incremental modification and optimization in VLSI Computer-Aided Design (CAD) is of fundamental importance. However, it has not been investigated as a discipline. Previous research and development effort is very unfocused and incomplete. 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. Full understanding and focused participation in research and development in the area of incremental and dynamic would help us cope with the complexity of present day VLSI systems and facilitates concurrent optimization. In this paper we formulate and survey fundamental problems in incremental 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 [26]. Fundamental issues such as wire-congestion and routability [71; 81], crosstalk and coupling noise [32; 79; 80], transmission-line behavior [26; 68], power consumption [12; 55], reliability and yield [41], and their interaction are crucial factors in designing next generation (VLSI) CAD tools. At the same time, time-to-market pressure is driving the electronic design automation strategists to reconsider design methodologies [9; 11; 22; 44; 60; 61].

Traditionally, CAD tools have been used more or less independently at the system-level, high-level, logic-level, and layout-level [58], 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 complicated with iterative design processes as faced in many complex projects requires very efficient incremental design algorithms and methodologies. Incremental algorithms are a must.

ISPD 2000, San Diego, CA.

There is a scattered literature in the area of incremental and dynamic data structures and algorithms. In the next 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 [76].

Incremental algorithms for synthesis and layout are needed when design undergoes local or incremental changes. 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 [8]. And efficient 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 portion of the design that we need to apply these algorithms to and what is the quality/speed tradeoff. A small number of research results in the area of incremental layout have been reported recently, focusing on floorplanning [25], placement [14], and FPGA routing [29; 65]. Incremental logic optimization has been studied in [8; 77], while [4] addresses BDD optimization using an incremental sifting algorithm. [78] presents an incremental approach for FSM traversal in the sense that the reachable states can be incrementally updated as local changes are made to the design. A recent work in [20] presents an incremental technology mapping algorithm for LUT-based FPGA that maintains the depth-optimality of the design. An incremental timing analysis algorithm is presented in [49]. Finally, a retiming-based incremental scheduling modification algorithm has been presented in [35].

In the next five subsections, we focus on formulations, possible solutions, and stating open problems in each area of incremental partitioning, floorplanning, placement, and routing.

2.1 Incremental and Dynamic Data Structures and Algorithms

Data structures are the backbone of all CAD tools. Efficient algorithms must be accompanied with an equally efficient data structure. Therefore, to facilitate incremental physical design, the underlying data structure must accommodate incremental changes.

Many of the VLSI CAD problems can be modeled as (hyper)graph or tree problems (see for example [58; 37; 50; 66; 69; 72; 75; 87].) Various incremental and dynamic algorithms have been proposed for (hyper)graphs and trees [37; 66; 75; 87]. It is therefore, worthwhile to study the incremental graph and tree algorithms and data structures [1; 5; 13; 30; 33; 64] in the context of VLSI CAD tools.

2.2 Partitioning

A chip may contain tens of millions of transistors. Layout of the entire circuit cannot be handled in a flat mode due to the limitation of memory space as well as computation power available. Even though fabrication technologies have made great improvements in packing more logic in a smaller area, the complexity of circuits has also been increasing correspondingly. This necessitates breaking a circuit and distributing it across several regions in a chip or across several chips. Thus, the first step in the physical design phase is partitioning. A good partitioning can significantly improve the circuit performance and reduce layout costs.

Partitioning is a complex and intractable problem that has been shown to be NP-complete. The nature of the partitioning problem along with the size of the circuit makes it difficult to perform an exhaustive search required to find an optimal solution. A number of effective heuristics have been proposed, e.g., [31; 40; 21; 73; 3; 23; 63; 39; 24; 6; 10]. Here, we focus our attention to incremental partitioning.

Incremental Partitioning: Given a hypergraph $H = (N, L)$, an incrementally changed hypergraph $H' = (N', L')$ can be constructed by adding/deleting vertices and/or edges from H. $N_{diff} = N \cap N'$ is the set of different vertices and $L_{diff} = L \cap L'$ is the set of different hyperedges between H and H'. In the context of incremental change, $|N_{diff}|$ and $|L_{diff}|$ is much smaller than $|N|$ and $|L|$. Given the original hypergraph H and a good partitioning solution of H, the problem of partitioning the incrementally changed hypergraph H' is called the incremental partitioning problem.

Very little work has been done on incremental partitioning. Here we present a simple approach to this problem. Since the incrementally changed hypergraph H' is only partially different than H, it is natural to think the partitioning solution on H is similar to the partitioning solution on H'. The difference in solutions of H and H' could be just the vertices in N_{diff} and the vertices close to them.

We define the value of *topological proximity* for each vertex in H' as follows: Vertices in N_{diff} and vertices corresponded to all the hyperedges in L_{diff} have a topological proximity value of 0 and are called 0-proximity vertices. Those remaining vertices which are directly connected to 0-proximity vertices have a proximity value of 1 and are called 1-proximity vertices. k-proximity vertices are those which are directly connected to at least one $(k-1)$-proximity vertex and no vertices with lower proximity value than $k-1$. The proximity values of all the vertices in N' can be quickly found by performing a breadth-first-search on H'.

A simple approach to solve the incremental partitioning problem is to fix most vertices with hight proximity values in their corresponding partitions as in the given good solution of H. The only active vertices capable of moving between partitions are those whose topological proximity value is smaller than a certain threshold. This threshold is called the *proximity threshold*. When the proximity threshold decreases, the number of active vertices decreases. Thus we spend less effort to solve the partitioning problem. When the threshold is zero, only 0-proximity vertices are possible to be moved around. Thus we can quickly find a partitioning solution for H'. However, this quick solution may not be good. When the threshold is infinity, all the vertices in H' are active. A better partitioning result and a longer running time are expected.

We tested this approach with different threshold value on five ISPD benchmark circuits [2]. For each benchmark circuit, we incrementally changed the netlist by 1%, 5%,10% and 20%, respectively. Table 1–4 shows the experimental results.

	threshold = 0			threshold = 1		
Ckt	act. vtx	cut	time	act. vtx	cut	time
ibm01	127	291	.11s	1384	291	.55
ibm02	196	285	.15s	2501	286	1.1
ibm03	231	767	.18s	2421	766	1.5
ibm04	275	617	.24s	2717	615	1.1
ibm05	293	1845	.28s	4269	1841	2.5
	threshold = 2			threshold = ∞		
Ckt	act. vtx	cut	time	act. vtx	cut	time
ibm01	7846	290	5.2s	12752	315	10.8s
ibm02	15261	280	16.4s	19601	276	16.8s
ibm03	15679	819	18.4s	23136	780	24.5s
ibm04	17374	596	15.3s	27507	545	25.4s
ibm05	23962	1846	31.6s	29347	1855	39.3s

Table 1: Incremental partitioning results when the difference between H and H' is 1%. We only move active vertices in H'. With each threshold value, active vertices are those whose proximity value is less than or equal to the threshold value. The "act. vtx" column is the number of active vertices. "cut" and "time" shows the net-cut and running time.

Table 1–4 show that when the difference between H and H' is not much ($< 10\%$), our simple approach with threshold of zero actually produces good partitioning results using only little amount of time. In this scenario, making more vertices active does not improve the quality of results substantially. When the difference between H and H' is large ($> 10\%$), partitioning results obtained from the zero-threshold approach are not very good. An alternative approach should be used here.

2.3 Floorplanning

In the floorplanning phase, the macro cells have to be positioned on the layout surface in such a manner that no blocks overlap and that there is enough space left to complete the interconnections. The input to the floorplanning is a set of modules, a list of terminals (pins for interconnections) for each module and a netlist, which describes the connections between the terminals.

There exist many different approaches to the floorplanning

	threshold = 0			threshold = 1		
Ckt	act. vtx	cut	time	act. vtx	cut	time
ibm01	637	495	.42s	5024	489	2.8
ibm02	980	293	.57s	8777	294	8.1
ibm03	1156	884	2.1s	8826	917	9.7
ibm04	1375	1015	0.96s	10291	1008	7.3
ibm05	1467	2260	1.1s	13706	2249	16.3
	threshold = 2			threshold = ∞		
Ckt	act. vtx	cut	time	act. vtx	cut	time
ibm01	11669	486	9.5s	12752	392	12.2s
ibm02	18902	289	19.8s	19601	284	19.1s
ibm03	21864	849	25.5s	23136	807	23.6s
ibm04	25487	1011	31.4s	27507	1040	28.2s
ibm05	28124	2232	43.4s	29347	2227	43.8s

Table 2: Incremental partitioning results when the difference between H and H' is 5%.

	threshold = 0			threshold = 1		
Ckt	act. vtx	cut	time	act. vtx	cut	time
ibm01	1275	714	.82s	7570	715	5.9
ibm02	1960	299	1.4s	12238	302	14.7
ibm03	2313	824	4.8s	13399	832	17.7
ibm04	2750	1453	1.8s	15406	1443	13.9
ibm05	2934	2744	1.9s	19297	2692	28.9
	threshold = 2			threshold = ∞		
Ckt	act. vtx	cut	time	act. vtx	cut	time
ibm01	12350	684	12.5s	12752	463	11.7s
ibm02	19320	296	24.0s	19601	287	21.5s
ibm03	22664	743	27.0s	23136	748	23.5s
ibm04	26755	1408	29.9s	27507	1196	34.1s
ibm05	28828	2730	52.0s	29347	2683	59.5s

Table 3: Incremental partitioning results when the difference between H and H' is 10%.

	threshold = 0			threshold = 1		
Ckt	act. vtx	cut	time	act. vtx	cut	time
ibm01	2550	1104	1.3s	9755	1069	10.6
ibm02	3920	369	4.1s	15702	310	19.4
ibm03	4627	1115	7.4s	17549	891	24.8
ibm04	5501	2320	7.1s	20793	2242	31.9
ibm05	5869	3432	3.8s	23907	3299	41.4
	threshold = 2			threshold = ∞		
Ckt	act. vtx	cut	time	act. vtx	cut	time
ibm01	12551	701	11.9s	12752	647	12.4s
ibm02	19472	292	21.7s	19601	294	21.7s
ibm03	22923	782	27.5s	23136	793	27.4s
ibm04	27225	1383	40.2s	27507	1339	41.2s
ibm05	29074	3246	63.3s	29347	3262	65.6s

Table 4: Incremental partitioning results when the difference between H and H' is 20%.

problem. When the area of the floorplan is considered, the problem of choosing for each module the implementation which optimizes a given evaluation function is referred to as the *Floorplan Area Optimization Problem* [67]. Wimer et al. [83] describe a branch and bound approach for the floorplan sizing problem, i.e. finding an optimal combination of all possible layout-alternatives for all modules after finding the topological order of modules. While their algorithm is able to find the best solution for this problem, it is very time consuming, especially for real problem instances. Cohoon et al. [38] implemented a genetic algorithm for the whole floorplanning problem. Their algorithm makes use of estimates for the required routing space to ensure completion of the interconnections. Another more often used heuristic solution method for placement is Simulated Annealing [84; 46].

As designs get larger and more complex, the floorplanning process becomes slower. However, having a good floorplan is essential both to the later stages of the design and for estimating the quality metrics of the product. Furthermore, in the design process of a chip, there are lots of changes to the netlist and/or module sizes due to changes at different levels of the design (e.g., changes in the module library, synthesis moves, timing optimization, etc.), that require obtaining a new floorplan which reflects the updated netlist. For example, when a module in the library of the modules is debugged or redesigned for quality improvements, its area changes. Or, when we perform logic changes (gate duplication, buffer insertion, etc.), the netlist and modules' area change. These changes can be significant compared to the original netlist and module sizes.

Due to time-to-market constraints, it is no longer affordable to redo the time-consuming process of floorplanning and associated top-level routing after each of these changes. Hence, we should take new approaches to these problems. One such approach is *incremental floorplanning*.

Incremental Floorplanning: Given a slicing (or non-slicing) floorplan F and some small changes in the module sizes and/or netlist, generate a new floorplan F' by applying small changes to F. The process needs to be much faster than generating F' from scratch. Ideally, we would like F' to be as good as a floorplan F'' generated using the modified netlist by a complete traditional floorplanning method (e.g., Simulate Annealing). In practice though, F' would be inferior to F'', and the quality loss will increase as the incremental changes build up. To remedy the quality loss, we should use some criteria to estimate how far F' is from F'', and if the difference is more than a certain threshold, generate a new floorplan using the updated netlist from scratch.

To minimize the frequency of invoking a traditional floorplanning process, the initial floorplan should be "tolerant" to incremental changes. A tolerant floorplan is able to accommodate the most likely changes[1] without significant increase in area or other design quality metrics. Also, the incremental changes in the floorplan should be applied in a way that minimizes the need for repeating the floorplanning from scratch.

Bazargan et. al. [7] propose a method for generating "tolerant" floorplans to accommodate changes in the module library. The proposed method gets as input a list of module

[1] Research effort should also be put in finding ways to predict netlist and module area changes and how likely they are to occur.

	Vec3	Vec4	Vec5	Vec6
Bres	3	6	8	11
Heap	443	9	392	501
Clip	49	15	15	19

Table 5: Floorplanning invocation count ratio

dimensions and the probability that each dimension is used in the final design, and generates a floorplan and the probability distribution function (PDF) of its dimensions. The output floorplan is picked in a way that its width/height variance is minimum (intuitively, changes in the module dimensions would not change the final area of the floorplan dramatically).

Crenshaw [25] uses a greedy method to apply local changes on a slicing floorplan tree. The algorithm is applied to a high-level synthesis framework. It uses an area-only criteria to assess the need for invoking a traditional floorplanner. Table 5 illustrates the speedup gained by this method over a traditional method (applying the floorplanning algorithm after each synthesis move). The columns are different sets of incremental changes in the netlist (Vec3 through Vec6). Each entry shows speed-ups for different floorplan results.

2.4 Placement

The placement problem can be defined as follows. Given an electrical circuit consisting of modules with predefined input and output terminals that are 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, possibly under timing and other constraints. 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 faster, but at the risk of resulting in poor layouts (with the exception of recent algorithms in analytical placement). 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. More recent constructive placement algorithms, such as numerical optimization techniques [45; 28] and integer programming formulation [51] yield better layouts but require significantly more CPU time. One of the biggest challenge for placement tools is the rapid growing circuit size. A good placement algorithm has to find a good layout as quickly as possible.

Incremental Placement: Given an existing good placement with respect to a given metric (e.g., wirelength), modify the placement to improve it in other metrics (e.g., congestion or timing). It is also possible that we would want to improve the original metric in reaction to netlist changes (cell and net addition/deletion). Incremental placement is usually caused by adding cells/nets, reducing congestion or meeting timing constraints.

A congestion reduction technique is proposed in [54] as a post processing stage after traditional wirelength minimization stage. Congestion that accounts for routability is minimized in this process without perturbing other placement objectives, e.g., wirelength. Generally it is extremely hard to modify a given "good" placement to meet particular constraints without degrading previously minimized objectives. For instance, given a placement produced by wirelength optimization tool, it is unlikely that a timing improvement process can work without increasing total wirelength. Thus, heuristics that achieve trade-off between several objectives should be studied/designed.

Some preliminary studies will be discussed next. Two cells are in *topological proximity* if there exists a short path from one cell to another. If two cells are closely located on placement we say they are in *geometrical proximity*. In a wirelength optimized placement topological proximity and geometrical proximity correlate well, i.e., usually two cells are topologically proximate if they are geometrically proximate and vice versa.

Table 6 shows a simple timing related experiment of incremental placement 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 long 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 bounding 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 the 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.

2.5 Routing

The routing problem can be defined as follows. Given floorplanning and placement results, 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 [19; 18]. 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

circuit	#cells	bounding 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**	**bounding 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 6: Number of long nets and total wirelength increase in reducing bounding boxes length of long nets in a good placement, N/A means this can not be done by this greedy approach. Initial placements are produced by TimberWolf(without timing constraints)

wirelength. The global routing results are used to guide detailed routing to compute the exact routing implementation in each tile.

The **incremental routing problem** can be formulated as follows: Given an existing layout and a set of nets to be added or rerouted, find a new layout to accommodate these incremental routing changes with the minimal modification of the original layout. There many scenarios that such an incremental routing problem may occur. For example, the netlist might be incrementally updated by the designer or synthesis tools, and some nets might be determined to have timing/noise violation after parasitic extraction and timing analysis. In this case, obviously, re-do the routing for all the nets is *too* time-consuming. Moreover, an entirely different layout may will 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.

We partition the incremental routing problem 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 and 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 on exiting routes (without changing the floorplan and placement results).
- **Incremental Floorplan and Placement Update**. If rip-up and reroute fails to complete all the 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 in fact 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 present some of our initial solutions in each of the three problems. Open problems in each of the three stages are also presented to motivate future research on this topic.

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 abstracted routing graph, which 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 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, *tiles*, where the center-line of a path can pass through [70; 56; 53]. These tiles are defined by the boundary of the obstacles and stored using a corner-stitching data structure [62]. In the point-based approach, the routing area is populated with points where the center-line of a path can pass through [85; 59; 86]. A maze searching algorithm [48; 34; 74] 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 [16]). 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. 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 [85; 82]. The drawback of the minimal graph approach is that it requires very expensive pre-construction. 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 [36]. Zheng, etc. al., presented an implicit representation of the routing graph,

Ex.	Uniform	Non-Uniform Grid		Iroute	
	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 7: ECO test results: experimental results comparing implicit representation of non-uniform graph with explicit uniform grid graph and Iroute using seven ECO examples.

that is, their graph nodes are computed on-the-fly [86]. The underlying graph in their approach is an extension to the *Track Graph* [85]. Their implicit approach, although very efficient in representation, is costly in computing the graph nodes.

We propose a non-uniform grid graph (NUGG) with its implicit representation for the single net routing problem [15]. 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 with uniform gridded approach, is much smaller. What is more, the gridded 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" [42; 47] plus a second level interval tree [27] 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 as we apply this graph and the auxiliary data structures to the incremental routing routing problem [15] (called the ECO problem in that paper). 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, as shown in Table 7. 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.

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 [52; 43; 57]. 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 makes it easy to model the resources in the routing region and simplifies the operation to exchange the resources between nets. However, this assumption 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 find the rerouting in a gridless environment.

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 [57] and Silk [52];
- *Those allows temporary design-rule violation*, such as the "cross-and-touch" router [43]

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. Therefore, we proposed an *wire planning* algorithm which effectively takes exact gridless design rules, models the available routing resources accurately, plans for the incremental routes prior to single net routing and also re-plans 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. Such an algorithm was successfully implemented in [17]. The results, as shown in Table 8, show that the detailed routing system, compares to a net-by-net approach using an efficient gridless detailed router, 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.

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 is closely related the incremental floorplan and placement problems formulated in Sections 2.3 and 2.4. There is, however, usually additional degree of flexibility in the context of incremental routing – the rip-up and re-route engine may suggest several alternatives for increasing routing sources to complete the routing. In this case, the incremental floorplan and placement algorithms need

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 8: 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.").

to choose the right combination of routing regions to increase 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.

3. CONCLUSION

In this paper, we formulated the incremental physical design problems and surveyed existing solutions. After discussing some preliminary results, a number of fundamental open problems were suggested. We highlight the importance of incremental modification and optimization in VLSI CAD. It was argued that 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. Full understanding and efficient solutions in this area would help us significantly in coping with the rapid increase in design complexity of current and future VLSI systems and facilitates concurrent optimization.

4. ACKNOWLEDGEMENT

The authors wish to thank graduate students at Northwestern and UCLA who have contributed to this paper: Kiarash Barzagan, Jie Fang, Maogang Wang, and Xioajian Yang.
This project was supported in part by grants from NSF (MIP-95-27389 and MIP-9357582) and grants from Fujitsu Laboratories at America under the California MICRO Program.

5. REFERENCES

[1] P.K. Agarwal, D. Eppstein, and J. Matousek. "Dynamic Half-Space Reporting, Geometric Optimization, and Minimum Spanning Trees". In *Annual Symposium on Foundations of Computer Science*, 1992.

[2] C. Alpert. "The ISPD98 Circuit Benchmark Suite". In *International Symposium on Physical Design*, pages 18–25. ACM, April 1998.

[3] C. J. Alpert, L. W. Hagen, and A. B. Kahng. "A General Framework For Vertex Orderings, With Applications to Netlist Clustering". *IEEE Transactions on VLSI Systems*, 4(2):240–246, 1996.

[4] M. Bae, G.T. Dong, S.W. Yang, and H. Chang. "An Optimization Technique for BDD Based on Bidirectional Incremental Sifting". *Korean Information Science Society, Comput. Syst. Theory*, 25(9):1058–1066, September 1998.

[5] M. Barbehenn and S. Hutshinson. "Efficient search and Hierarchical Motion Planning by Dynamically Maintaining Single-Source Shortest Paths Trees". *IEEE Transactions on Robotics and Automation*, 11(2):198–214, April 1995.

[6] S. T. Barnard and H. D. Simon. "A Fast Multilevel Implementation of recursive Spectral Bisection For Partitioning Unstructured Problem". In *SIAM Conference on Parallel Processing for Scientific Computing*, pages 711–718, 1993.

[7] K. Bazargan, S. Kim, and M. Sarrafzadeh. "Nostradamus: A Floorplanner of Uncertain Designs". *IEEE Transactions on Computer Aided Design*, 18(4):389–397, April 1999.

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

[9] R.G. Bushroe, S. DasGupta, A. Dengi, P. Fisher, S. Grout, G. Ledenbach, N. Nagaraj, and R. Steele. "Chip Hierarchical Design System (CHDS): A Foundation for Timing-Driven Physical Design into the 21st Century". In *International Symposium on Physical Design*, pages 212–217, 1997.

[10] A. E. Caldwell, A. B. Kahng, and I. L. Markov. "Improved Algorithms For Hypergraph Bipartitioning". In *Asia and South Pacific Design Automation Conference*. IEEE/ACM, 2000.

[11] R. Camposano. "The Quarter-Micron Challenge: Integrating Physical and Logic Design". In *International Symposium on Physical Design*, page 211, 1998.

[12] A. Chandrakasan and R. Brodersen. "Minimizing Power Consumption in Digital CMOS Circuits". *Proceedings of the IEEE*, pages 498–523, April 1995.

[13] Y.-J. Chiang and R. Tamassia. "Dynamic Algorithms in Computational Geometry". *Proceedings of the IEEE*, 80(9):1412–1434, September 1992.

[14] 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.

[15] 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.

[16] 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.

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

[18] J. Cong and L. He. Theory and algorithm of local-refinement-based optimization with application to device and interconnect sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(4):406–420, Apr 1999.

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

[20] J. Cong and H. Huang. Technology mapping for field programmable gate arrays with incremental changes. In *Proc. 37th Design Automation Conference*, page to be appeared, Jun 2000.

[21] J. Cong, P. Li, L. Sung, T. Shibuya, and D. Xu. Large scale circuit partitioning with loose/stable net removal and signal flow based clustering. In *Proc. IEEE International Conference on Computer-Aided Design*, pages 441–446, Nov 1997.

[22] J. Cong and D.Z. Pan. "Interconnect Estimation and Planning for Deep Submicron Designs". In *Design Automation Conference*, pages 507–510, 1999.

[23] J. Cong and M. L. Smith. "A Parallel Bottom-up Clustering Algorithm With Applications to Circuit Partitioning in VLSI Design". In *IEEE-ACM Design Automation Conference*, pages 755–760, 1993.

[24] J. Cong and L. Sung. Multiway partitioning with pairwise movement. In *Proc. IEEE International Conference on Computer-Aided Design*, pages 512–516, Nov 1998.

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

[26] A. Deutsch, G.V. Kopcsay, P.J. Restle, H.H. Smith, G. Katopis, W.D. Becker, P.V. Coteus, C.W. Surovic, B.J. Rubin, R.P. Dunne Jr., T. Gallo, K.A. Jenkins, L.M. Terman, R.H. Dennard, G.A. Sai-Halasz, B.L. Krauter, and D.R. Knebel. "When are Transmission-Line Effects Important for On-Chip Interconnections?". *IEEE Transactions on Microwave Theory and Techniques*, 45(10):1836–1846, October 1997.

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

[28] H. Eisenmann and F. Johannes. Generic global placement and floorplanning. In *Proc. 35th Design Automation Conference*, pages 269–274, Jun 1998.

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

[30] D. Eppstein. "Sparsification - A Technique for Speeding up Dynamic Graph Algorithms". In *Annual Symposium on Foundations of Computer Science*, 1992.

[31] C. M. Fiduccia and R. M. Matteyses. "A Linear Time Heuristic for Improving Network Partitions". In *Design Automation Conference*, pages 175–181, 1982.

[32] J.-Y. Fourniols, M. Rocca, F. Caignet, and E. Sicard. "Characterization of Crosstalk Noise in Submicron CMOS Integrated Circuits". *IEEE Transactions on Electromagnetic Compatibility*, pages 271–280, August 1998.

[33] M. Goodrich and R. Tamassia. "Dynamic Trees and Dynamic Point Locations". *SIAM Journal on Computing*, 28(2), February 1998.

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

[35] S. Hassoun. "Fine Grain Incremental Rescheduling via Architectural Retiming". In *International Symposium on System Synthesis*, pages 158–163. IEEE, 1998.

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

[37] K.C. Ho and S.B.K. Vridhula. "Interval Graph Algorithms for Two-Dimensional Multiple Folding of Array-Based VLSI Layouts". *IEEE Transactions on Computer Aided Design*, 13(10):1201–1222, October 1994.

[38] J.P.Cohoon. "Distributed Genetic Algorithms for the Floorplan Design Problem". *IEEE Transactions on Computer Aided Design*, 10(4):483–492, April 1991.

[39] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. "Multilevel Hypergraph Partitioning: Application in VLSI Domain". In *IEEE-ACM Design Automation Conference*, pages 526–529. IEEE/ACM, 1997.

[40] G. Karypis and V. Kumar. "METIS 3.0: Unstructured Graph Partitioning and Sparse Matrix Ordering System". In *Technical Report 97-061*. Department of Computer Science, University of Minnesota, 1997.

[41] G. Karypis and V. Kumar. "Multilevel k-way Hypergraph Partitioning". In *IEEE-ACM Design Automation Conference*, pages 343–348, 1999.

[42] 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.

[43] 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.

[44] K. Keutzer, A. R. Newton, and N. Shenoy. "The Future of Logic Synthesis and Physical Design in Deep Submicron Process Technologies". In *International Symposium on Physical Design*, pages 218–224, 1997.

[45] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization". *IEEE Transactions on Computer Aided Design*, 10(3):365–365, 1991.

[46] R. M. Kling and P. Banerjee. "Optimization by Simulated Evolution with Applications to Standard Cell Placement". In *Design Automation Conference*, pages 20–25, 1990.

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

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

[49] J.-F. Lee and D.T. Tang. "An Algorithm for Incremental Timing Analysis". In *Design Automation Conference*, pages 696–701. ACM/IEEE, 1995.

[50] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, 1990.

[51] T. Lengauer and M. Lugering. "Integer Programming Formulation of Global Routing and Placement Problems". *World Sceintific*, 1993. Special volume on *Algorithm Aspects of VLSI Layout*, (M. Sarrafzadeh and D. T. Lee eds.).

[52] 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.

[53] 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.

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

[55] E. Macii, M. Pedram, and F. Somenzi. "High-Level Power Modeling, Estimation, and Optimization". *IEEE Transactions on Computer Aided Design*, pages 1061–1079, November 1998.

[56] 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.

[57] 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.

[58] G. De Micheli. *"Synthesis and Optimization of Digital Circuits"*. McGraw-Hill, New York, 1994.

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

[60] R.H.J.M. Otten. "Global Wires Harmful?". In *International Symposium on Physical Design*, pages 104–109, 1998.

[61] R.H.J.M. Otten and R.K. Brayton. "Planning for Performance". In *Design Automation Conference*, pages 122–127, 1998.

[62] 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.

[63] A. Pothen, H. D. Simon, and K. P. Liou. "Partitioning Sparse Matrices With Eigenventors of Graphs". *SIAM Journal of Matrix Analysis and Applications*, 11(3):430–452, 1990.

[64] G. Ramalingam. *"Bounded Incremental Computation, Lecture Notes in Computer Science, Volume 1089"*. Springer, 1996.

[65] 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.

[66] C.P. Ravikumar, R. Aggarwal, and C. Sharma. "A Graph Theoretic Approach for Register-File Based Designs". In *International Conference on VLSI Design*, 1979.

[67] M. Rebaudengo and M. S. Reorda. "GALLO: A Genetic Algorithm for Floorplan Area Optimization". *IEEE Transactions on Computer Aided Design*, 15, 1996.

[68] P.J. Restle, K.A. Jenkins, A. Deutsch, and P.W. Cook. "Measurement and Modeling of On-Chip Transmission Line Effects in a 400 Mhz Microprocessor". *IEEE Journal of Solid-State Circuits*, pages 662–665, April 1998.

[69] M. Sarrafzadeh and C.K. Wong. *"An Introduction to VLSI Physical Design"*. McGraw-Hill Book Company, 1996.

[70] 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.

[71] H.-F. S.Chen and D.T. Lee. "On Crossing Minimization Problem". *IEEE Transactions on Computer Aided Design*, pages 406–418, May 1998.

[72] N. Sherwani. *"Algorithms for VLSI Physical Design Automation"*. Kluwer Academic Publishers, 1997.

[73] H. Shin and C. Kim. "A Simple Yet Effective Technique For Partitioning". *IEEE Transactions on VLSI Systems*, 1(3), 1993.

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

[75] D.L. Springer and D.E. Thomas. "Exploiting the Special Structure Conflict and Compatibility Graphs in high-Level Synthesis". *IEEE Transactions on Computer Aided Design*, 13(7):843–856, July 1994.

[76] 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.

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

[78] G.M. Swamy, R.K. Brayton, and V. Singhal. "Incremental Methods for FSM Traversal". In *International Conference on Computer Design*, pages 590–595, 1995.

[79] K.T. Tang and E.G. Friedman. "Interconnect Coupling Noise in CMOS VLSI Circuits". In *International Symposium on Physical Design*, pages 48–53, 1999.

[80] A. Vittal and M. Marek-Sadowska. "Crosstalk Reduction for VLSI". *IEEE Transactions on Computer Aided Design*, pages 290–298, March 1997.

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

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

[83] S. Wimer, I. Koren, and I. Cederbaum. "Optimal Aspect Ratios of Building Blocks in VLSI". *IEEE Transactions on Computer Aided Design*, 8(2):139–145, 1989.

[84] D. F. Wong, H. W. Leong, and C. L. Liu. *Simulated Annealing for VLSI Design*. Kluwer Academic, 1988.

[85] 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.

[86] 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.

[87] J. Zhu and M. Abd-El-Barr. "On the Optimization of CMOS Circuits". *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 40(6):412–422, June 1993.