Sidewinder - A Scalable ILP-Based Router

Jin Hu, Jarrod A. Roy, and Igor L. Markov
The University of Michigan, Department of EECS
2260 Hayward Ave., Ann Arbor, MI 48109-2121
{jinhu, royj, imarkov}@eecs.umich.edu

ABSTRACT

We propose Sidewinder, a new global router that combines pattern routing and maze routing in a novel, incremental, ILP formulation. It is the first flat ILP-based approach scalable enough to consider over 10^4 GCells at once. Moreover, it also can be used as a component in previously proposed multi-level and progressive ILP schemes. Sidewinder is particularly good at finding routes with minimal via count, which can improve yield in sub-90nm technologies. Other innovations in our work include an ILP construction based on a dynamically-updated congestion map and the use of C-shape routes to alleviate local congestion and improve routability. On well-known benchmarks, Sidewinder improves routed wirelength and reduces via count by over 6% compared to ILP-based BoxRouter 1.0 and 35.8% compared to DLM-based FGR 1.0. This easy-to-implement methodology is extensible to detail routing of ASICs as well as FPGAs where it can account for complex design rules and models.

Categories and Subject Descriptors

J.6 [Computer-Aided Engineering]: Computer-aided design(CAD)

General Terms

Algorithms, Performance, Design

Keywords

Integer Linear Programming, Global Routing

1. INTRODUCTION

Routing is one of the key steps in the VLSI design flow process, as it impacts circuit performance, power, and production. Routing directly affects the timing of the design, as the process determines length of the critical paths. Traditionally, a router's main goal is to only minimize wirelength. However, with the current technology scaling trends, designs are susceptible to coupling capacitance and other parasitic effects. Traversing from one metal layer to another is becoming costly as now vias have non-trivial effects because they significantly impact timing and may block several routing tracks [18]. In this respect, routing is even more important, as it directly determines the locations and number of vias. Thus, a router must also limit the number of vias as well as minimize the wirelength of a given design.

The problem of (global) routing is known to be NP-Complete [12]. Thus, there are two main approaches to handling this issue: heuristics and (integer) linear programming (ILP). Almost all current academic routers [3,7,15] are based on the former, the main reason being that the latter

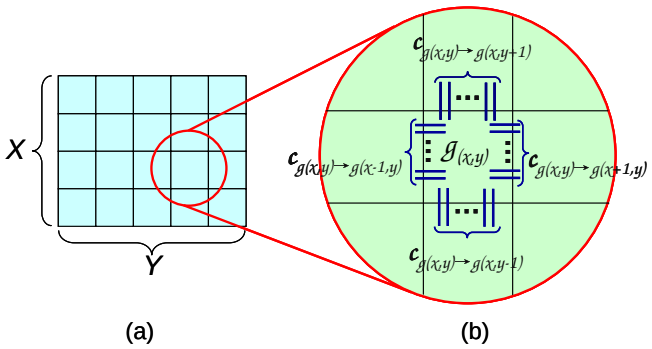


Figure 1: A design's routing grid. (a) A sample routing grid (5x5) for a generic design. (b) A close-up of a GCell and the edge capacities in each direction. The routing grid abstraction models a multi-metal layer problem where each layer can be connected to another layer with the use of vias.

lacks scalability. The first ILP-based router was proposed by Burstein and Pelavin [2] but was impractical because ILP solvers of the day were unacceptably slow. ILP solvers have improved dramatically in terms of speed and efficiency in the past twenty years and very recently M. Cho and D. Pan (BoxRouter 1.0) [3] have successfully implemented an ILP-based router with pre- and post-processing to simplify the problem.

In this paper, we propose Sidewinder, an iterative, congestion-driven ILP-based advanced pattern router. Like BoxRouter 1.0, we consider routing optimally all two-pin nets with L shapes first. However, instead of iteratively expanding a small bounding box, we consider the entire routing grid during each pass. In addition to L shapes, we also consider all Z shapes and selected C shapes (i.e. slightly detoured routes). In the ILP formulation, each net has only two allowed patterns, which are chosen based on a congestion map. On the ISPD98 benchmarks [10], this formulation alone routes 98% of nets with optimal wirelength and minimal via count, but remaining nets require small detours.

Sidewinder is much simpler than existing routers because the majority of work is done by the ILP solver. Unlike the ILP formulations used in BoxRouter 1.0 [3], Sidewinder's pattern routes allow at most three bends per two-pin subnet and detours of at most four GCells in length. With these restrictions relaxed, any remaining nets can be routed with a simple post-routing step in all the designs we considered. On the other hand, Figure 2 suggests that Sidewinder is already a viable global router because post-processing can be

performed by existing detail routers. Sidewinder's ILP formulation can also be used in the BoxRouter flow to improve via count and detours.

The following key ideas are proposed in this work:

- the selection of two least congested patterns per net
- search over all 2-bend Z-shaped routes
- the use of slightly detoured 2-bend and 3-bend C-shape routes
- congestion-based ILP formulation
- congestion map updates between ILP calls
- an incremental ILP for all nets that is guaranteed to never make solutions worse

The rest of the paper is structured as follows: Section 2 states background information and previous work. Section 3 describes the problem formulation in detail. Section 4 has the experimental setup and results. Finally, Section 5 concludes the paper and mentions future work.

2. BACKGROUND AND PREVIOUS WORK

Below we review global routing, detail routing and several known routing approaches.

2.1 Global and Detail Routing

Routing is typically split into two stages: global and detail routing. During global routing, design rules are not considered and the design itself is broken down into a grid made up of global routing cells or GCells (see Figure 1). Each net is then assigned routing segments within each GCell such that the terminal pins are connected. To model routing resources, capacities are assigned to each GCell edge to limit the number of routing segment assignments. A global routing solution is considered legal if all nets are connected and all edge capacities are respected.

After global routing, detail routing is applied to the design. Given the routing segments assigned to each GCell from global routing, detail routing physically assigns each segment to specific routing tracks. Detail routing also takes into account spacing rules specific to the design.

Note that a purely legal global routing solution is not required for the detail router, as illustrated in Figure 2, an excerpt from a Cadence WarpRoute report on a test benchmark. It shows that although global routing reported 295 GCells with violations, the detail routing solution is legal. As long as the percentage of violations is small, detail routing is usually able to compensate.

Recently developed routers include work from Hadsell and Madden (Fengshui with *Chi* dispersion) [7], M. Cho and D. Pan (BoxRouter 1.0) [3], Roy and Markov (FGR 1.0) [17], as well as M. Pan and C. Chu (FastRoute) [15, 16]. Fengshui, BoxRouter 1.0, and FGR 1.0 minimize total routed wirelength, while FastRoute minimizes its runtime at the cost of higher wirelength.

2.2 Routing Approaches

The following discusses traditional as well some recent approaches to global routing.

Pattern Routing. Pattern routing significantly reduces the problem's solution space and improves runtime. Instead of having restrictions placed on each routing segment, each net is limited to a small number of shapes. A two-pin net is commonly mapped to an L shape, where only one bend is used and the wirelength is optimal, or a Z shape, where two horizontal segments are connected with a middle vertical segment or vice versa. Kastner et al. [13] have shown that in standard application specific integrated circuits (ASICs), pattern routing is efficient, as it minimizes via count and increases scalability. Further work done by Westra et al. by [19] shows that in ASIC routing, the majority of two-pin nets can be routed using only L shapes. Typically, pattern routing is more flexible than routing only Ls and Zs - each pattern is selected from a finite set of routing topologies.

	Cadence WarpRoute Report		
(1)	Total wire length	=	6270421
(2)	Total number of vias	=	740208
(3)	Total number of violations	=	0
(4)	Total number of over capacity gcells	=	295 (0.07%)
(5)	Total CPU time used	=	0:30:36
(6)	Total real time used	=	0:30:36
(7)	Maximum memory used	=	162.00 megs

Figure 2: Excerpt from Cadence WarpRoute on a test benchmark. Notice that although global routing produced a total of 295 GCells with violations (line 4), the final result given by detail routing (line 3) has none. This is typical for dozens of industry circuits we routed.

Maze Routing. The most common routing technique used today, maze routing uses standard search algorithms such as BFS and Dijkstra's algorithm [6] to connect terminals along the routing grid. While shortest routed lengths are found for pairs of terminals, the order in which nets are routed has a profound effect on solution quality and routed length. As a result, maze routing must be applied many times with heuristic net orderings to find legal solutions. In addition, vias must be modeled explicitly to prevent unnecessary detouring.

SAT- and ILP-based Routing. Modeling routing constraints by Boolean formulas in CNF, Nam et al. [14] developed a SAT-based detail router which routes all nets simultaneously. Using ILP, this formulation can be extended to route as many nets as possible [20]. ILP-based routing has traditionally been avoided due to its lack of scalability. An early attempt by Burstein and Pelavin [2] could not be efficiently implemented because contemporary ILP solvers were not sufficiently powerful. However, after major advancements in ILP solvers, the idea of routing optimally using ILP became an option. Recently, M. Cho and D. Pan developed BoxRouter 1.0 [3]. After decomposing multi-pin nets into two-pins subnets, BoxRouter 1.0 uses pattern routing and begins at the most congested region. Starting within a small bounding box, it optimally routes as many nets in the region as possible using only L patterns; the remaining unrouted nets are given to a maze router. The bounding box is iteratively expanded using a *progressive* ILP formulation that extends partially-routed nets with additional L-shaped segments. Then maze routing is invoked to complete nets that did not route. Such steps are repeated until the entire global routing grid is subsumed. Given that ILPs are solved optimally, using powerful ILP solvers can only improve runtime. However, a faster ILP solver may facilitate a more comprehensive ILP formulation.

One common method used to improve the scalability of

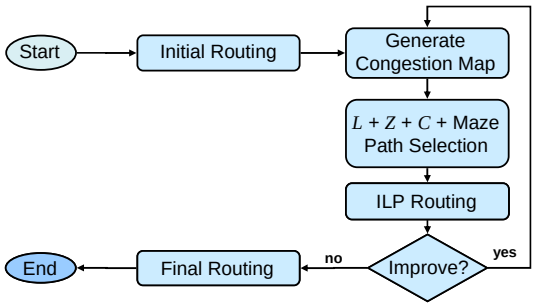


Figure 3: High-level flow of Sidewinder. We first create an initial solution using only L shapes. Next, we build a congestion map based on the current solution to use as a guide for the new solution. For net path candidates, we consider Ls, Zs, Cs, and a maze route. Once all nets are processed, an ILP is formed and solved. This cycle continues until the new solution has the same cost as the current solution. Once there is no more improvement, maze routing is applied to yield the final routing solution.

ILP-based routing techniques is to relax the ILP problem into an easier linear programming (LP) problem. Multi-commodity flow (MCF) based routers take this approach [1, 8]. An approximation technique incrementally adjusts routing edge weights and builds new Steiner tree topologies for each net at every iteration to solve the LP. BoxRouter 1.0 has been compared to a recent MCF-based router and was found to be superior in speed and solution quality [3].

3. SIDEWINDER

Below we introduce our approach to global routing, starting from the problem framework. We then present the high-level flow, related algorithms, and our ILP formulation.

3.1 Problem Framework

For every design, a finite global routing grid G of size $X \times Y$ is defined. For simplicity, we define the bottom left GCell of G to be (0,0) and only consider rectilinear paths. For an arbitrary GCell $g(x, y)$, we define four edge capacities $c_{g(x,y)\rightarrow g(x+1,y)}, c_{g(x,y)\rightarrow g(x-1,y)}, c_{g(x,y)\rightarrow g(x,y+1)}$, and $c_{g(x,y)\rightarrow g(x,y-1)}$, one for each cardinal direction, as shown in Figure 1. Finally, we only consider the routing of two-pin nets; multi-pin nets are decomposed into multiple two-pin nets. The terminals of each net are located within their respective GCells.

3.2 High-Level Framework

As shown in Figure 3, we first generate an initial routing solution using only L shapes (initial routing). Using this current solution, we build a congestion map to guide the routing of the new solution. For each net, we consider Ls, Zs, Cs, and a maze route as possible path candidates. This specific portion is discussed in greater detail in Algorithm 1, Algorithm 2, and Section 3.3. After all the path candidates have been selected, we formulate this problem into an ILP and generate the new routing solution. If this new solution is better (higher objective function) than the previous solution, this process is repeated. Once there is no more improvement, we apply a final round of maze routing to route all outstanding nets.

Algorithm 1: High-Level Iterative Algorithm of Sidewinder

Input: Routing Grid ***G***, Netlist ***N***, Solution ***R***

```
 1:  CurSol = R;
 2:  improvement = INT_MAX;
 3:  while (improvement > 0)
 4:       MakeCongestionMap(G, CurSol);
 5:       for each unrouted net n in N
 6:            pqueue pq = 0;
 7:            for each path type PT
 8:                 pq.push(CreatePath(PT));
 9:            pq.pop(n.path1);
10:            pq.pop(n.path2);
11:       for each routed net n in N
12:            pqueue pq = 0;
13:            for each path type PT
14:                 pq.push(CreatePath(PT));
15:            pq.pop(n.path1);
16:            n.path2 = n.currentPath;
17:       Inst = MakeILP(G, N);
18:       NewSol = SolveILP(Inst);
19:       improvement = SQ(NewSol) - SQ(CurSol);
20:       CurSol = NewSol;
```

Output: NewSol

Figure 4: High-level algorithm of Sidewinder. The first iteration routes as many subnets as possible using Ls. In subsequent iterations, alternative paths of Ls, Zs, Cs, and a maze route are evaluated using a congestion map.

3.3 Algorithm Design

The iterative portion of Sidewinder is given in Figure 4. Based on an initial routing solution R, we place all routed nets to construct an initial congestion map. As noted by line 16, all routed paths will be used as one possible path candidate in the next iteration. The alternate candidate will be the best (least congested) path out of: all possible Ls, all possible Zs, all possible Cs, and a route generated by a maze router (pattern routes are depicted in Figure 5). For each unrouted net, as there is no current path to use, the two best paths will be selected as candidates from the aforementioned list. To improve routability, we evaluate all unrouted nets before routed nets.

For each net, we only consider legal path candidates, e.g., detoured paths that are not within the routing grid are not allowed. Each of the shapes are also considered "sufficiently different" — this gives the router more flexibility and freedom. We emphasize that the two chosen paths are always different. In the case where the maze route is a duplicate pattern path, the maze route is removed and the next best path comes off the priority queue.

Once the two routes are selected, the congestion map is updated. If the net was routed, the current path is given a weight of 0.9 and the new candidate 0.1. If the net was not routed, each candidate is given a weight of 0.5. Notice that the congestion map is updated after each net has been processed. This guides the router such that the new path choices will not create new congestion areas.

After each net has two possible path candidates, we create the ILP formulation and solve. This yields a new routing solution $NewSol$. If the solution quality of $NewSol$ is better (more than) than the solution quality of R, then $R = CurSol$ and the process is repeated. From our formulation, we define the quality of a routing solution to be the objective function returned by the ILP solver. A higher objective value implies more nets have been routed. Once the objective value stabilizes, i.e., SQ(NewSol) = SQ(CurSol), the iterative portion of Sidewinder terminates.

The algorithm for path calculation and selection is given in Figure 6. Each candidate route is given two metrics: minimum number of free segments (*pathFree*) and total number of free segments (*pathTotal*). *pathFree* is found by taking the minimum available space/segment for each segment in the path. If a segment has no room (capacity = 0) or is overfilled (capacity < 0), the priority is the -(total number of routing violations). In other words, paths with overflow have a negative priority (less desirable) while paths without any violations have a positive priority (more desirable). Likewise, *pathTotal* is found by summing up the total number of free space across the path.

Once all the path priorities are calculated, they are ranked by *pathFree*. That is, the least congested paths are the top choices while the most congested paths are at the bottom. *pathTotal* is only used in case of a tie between paths that have the same *pathFree*. Thus, the most desirable path is the one with the most total available capacity. Note that with this formulation, there are ALWAYS at least two legal and "sufficiently different" paths available.

With this formulation, we *guarantee that the ILP solution will be no worse than the previous.* Each subsequent ILP instance routes at least as many nets as the current ILP instance. In the worse case, the same nets will be routed, causing the objective function to stay constant.

3.4 ILP Formulation

We present the general case of our ILP formulation. Note that the first ILP iteration is a special case where the two selected paths are *L*s. Variables are explained after the formulation.

Maximize:

$$\sum_{n} w_{1n}x_{1n} + w_{2n}x_{2n}$$

Subject to:

$$x_{1n} + x_{2n} \leq 1 \qquad \forall\ n$$

$$x_{1n} \in \{0,1\} \qquad \forall\ n$$

$$x_{2n} \in \{0,1\} \qquad \forall\ n$$

$$\sum_{\substack{x_{1n},x_{2n} \\ \in g(i,j)\rightarrow g(i,j+1)}} x_{1n} + x_{2n} \leq c_{g(i,j)\rightarrow g(i,j+1)} \qquad 0 \leq i < X, 0 \leq j < Y-1$$

$$\sum_{\substack{x_{1n},x_{2n} \\ \in g(i,j)\rightarrow g(i,j-1)}} x_{1n} + x_{2n} \leq c_{g(i,j)\rightarrow g(i,j-1)} \qquad 0 \leq i < X, 0 < j < Y$$

$$\sum_{\substack{x_{1n},x_{2n} \\ \in g(i,j)\rightarrow g(i+1,j)}} x_{1n} + x_{2n} \leq c_{g(i,j)\rightarrow g(i+1,j)} \qquad 0 \leq i < X-1, 0 \leq j < Y$$

$$\sum_{\substack{x_{1n},x_{2n} \\ \in g(i,j)\rightarrow g(i-1,j)}} x_{1n} + x_{2n} \leq c_{g(i,j)\rightarrow g(i-1,j)} \qquad 0 < i < X, 0 \leq j < Y$$

Where:

N	:	netlist
G	:	routing grid
$X \times Y$	:	size of G
x_{1n}, x_{2n}	:	two paths for each net $n \in N$
w_{1n}, w_{2n}	:	net weights for x_{1n}, x_{2n}
$g(i,j)$	:	grid cell $\in G$
$c_{g(i,j)\rightarrow g(i\pm1,j\pm1)}$	:	capacities

Recall that we choose two possible path candidates for each net n in the netlist N. In the ILP, this is represented with two 0-1 variables, x_{1n} and x_{2n}. A value of 0 represents the path was not chosen; the value of 1 represents the path chosen for the net. The first three constraints guarantees that at most one path out of the two will be selected (either one path will be chosen or no paths will be chosen). The fourth constraints states that for all *North* routing edges $g(i,j) \rightarrow g(i,j+1) \in G$, the summation of all taken paths must be less than or equal to $c_{g(i,j)\rightarrow g(i,j+1)}$, the total capacity of $g(i,j)$. That is, the sum of routing segments assigned through a GCell must be less than or equal to the total capacity of the edge. Similarly, the next three constraints ensure that *South*, *East*, and *West* edge capacities are respected. Note that only the *North* and *East* (or some similar variation) constraints are needed, as the *North* and *South* constraints are the same and the *East* and *West* are the same. Lastly, the variables w_1 and w_2 are the corresponding weights given to each path. These weights are determined by the type of path x_{1n} and x_{2n} are. Strictly speaking, a higher coefficient implies that path is more preferred than a path with a lower coefficient.

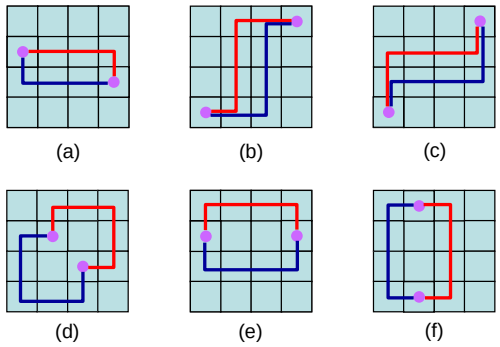


Figure 5: Patterns Sidewinder considers when choosing paths. (a) Two different *L* shapes, (b) All possible vertical *Z*s, (c) All possible horizontal *Z*s, (d) *C* shapes - detouring one unit in the vertical direction, (e) *C* shapes - detouring one unit in the horizontal direction, (f) *C* shapes - detouring one unit in both the horizontal and vertical direction.

Since we consider a number of paths with different wirelength and bends (an *L* has less wirelength and fewer bends than a detour), we assign different weights to the objective function based on the type of path selected. Since the objective function is maximized, we value *L*s the most, followed by *Z*s, then *C*s, and finally maze routes. Note that although we consider many different paths, the number of variables needed is still only two per subnet, ensuring the scalability of our ILP formulation.

3.5 Insights

During our preliminary work, we have evaluated a number of different ILP formulations to global routing. We quickly observed that all formulations that scale to a large number of nets fell into the category of pattern routing. That is, they would only allow a small number of configurations per net. Furthermore, ILP formulations with only two patterns per net were solved an order of magnitude faster than those with four or more patterns per net.

While our observations about efficient ILP formulations are consistent with the success of *L*-shape routing in BoxRouter 1.0, the choice of *L*-shapes is not as critical. Thus our first insight is as follows: *Select routing patterns other than L-shapes for nets and allow for dynamic selection of*

```
Algorithm 2: Path Selection
Input: Routing Grid G, Path p
      1:  pathFree = 0;
      2:  pathTotal = 0;
      3:  if (p.illegal)
      4:       pathFree = PATH_ILLEGAL;
      5:       pathTotal = PATH_ILLEGAL;
      6:  for each segment s in p
      7:       if (G[s].capacity < 0)
      8:            if (pathFree >= 0)
      9:                 pathFree = -1;
     10:            else
     11:                 pathFree--;
     12:       else
     13:            pathFree = min(pathFree, G[s].capacity);
     14:            pathTotal += G[s].capacity);
Outputs: pathFree, pathTotal
```

Figure 6: Algorithm to find the least congested path. The path priority is based on the minimum number of free segments along the path ($pathFree$) and the total number of free segments ($pathTotal$). All path choices are ranked in ascending order first with respect to $pathFree$ and then $pathTotal$. Thus, the dominant metric for congestion is $pathFree$ and $pathTotal$ is only used as tie-breaking criterion.

pattern shapes.

For further studies, we extracted several small but difficult routing instances from common benchmarks. In some of the instances, only about half the nets could be routed with Ls due to capacity constraints. We have evaluated several simple patterns, including Z-shapes where the middle segment would cross the midpoint of the net's bounding box. We found that allowing this pattern provides only marginal (if any) improvement to L-only ILPs. However, *including shapes with slight detours (which we term as C-shapes) allowed us to route significantly more nets.*

Our third insight is *paths should be evaluated using a congestion map, rather than strictly length or via count, to determine the best candidates.* For the initial ILP formulation, we select the two best paths based on congestion if the net was not routed previously and the current and best paths if the net was routed. We noticed that the runtime of the ILP solver decreased dramatically the more accurate we were at predicting the possible paths.

Our final insight is that *all Z-shaped paths should be considered rather than only ones that cross the midpoint of a nets' bounding box.* For a given net, we can scan the congestion map and find quickly the least congested Z-shaped paths. We noticed that this new flexibility noticeably improved our solution quality.

3.6 Sidewinder vs. BoxRouter 1.0

Comparing our ILP formulation with BoxRouter 1.0 — the only scalable ILP router in the literature — we note several important differences:

- BoxRouter's ILP is applied to a small region and includes only L-shaped routes; our formulation is applied to the entire global routing grid and after the first iteration also includes all possible C-shapes and Z-shapes.
- For long nets, BoxRouter's ILP routes one portion of the net at a time, whereas Sidewinder's ILP routes entire nets in all cases.
- At each iteration, BoxRouter's *progressive* ILP extends its *current region* to a slightly larger region and extends nets present in both regions by new L-shaped segments. Therefore, long nets may be routed with two bends per region[1], whereas the baseline Sidewinder formulation is *global* and does not allow more than three bends per subnet.
- BoxRouter's ILP formulation is not sensitive to congestion, but is formulated for the most congested region in its first iteration. In contrast, Sidewinder's ILP formulation is global. The second iteration (and beyond) explicitly accounts for congestion when selecting two patterns for each net. Moreover, the status of the internal congestion map is dynamically updated during the ILP construction.

Benchmark	Grid	Total nets	Nets routed	Iterations	Runtime (min)
ibm01	64×64	11507	99.36%	12	231
ibm02	80×64	18429	99.95%	8	92
ibm03	80×64	21621	99.99%	6	93
ibm04	96×64	26163	99.50%	6	217
ibm05	128×64	27777	100%	1	< 1
ibm06	128×64	33354	99.98%	6	130
ibm07	192×64	44394	99.94%	6	100
ibm08	192×64	47944	99.98%	6	120
ibm09	256×64	50393	99.99%	6	277
ibm10	256×64	64227	99.98%	5	103
Average			99.86%		

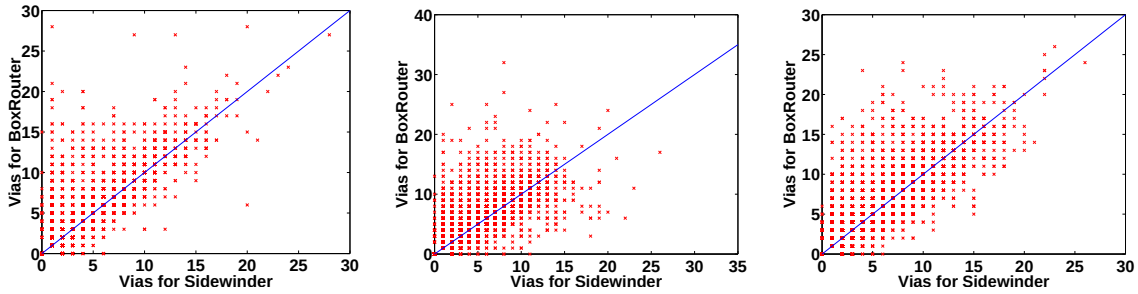
Table 1: Results of routability for Sidewinder on the ISPD98 benchmark suite [10] BEFORE FINAL ROUTING.

4. EMPIRICAL VALIDATION

We implemented Sidewinder as follows. The high-level algorithms are written in C++; we used CPLEX v.10.1 [9] as our ILP solver. Using FLUTE [5], we decompose all multi-pin nets into two-pin subnets. For our ILP cost function, we use the following pricing scheme for the different patterns: 1.00 for Ls, 0.99 for Zs, 0.98 for Cs, and 0.97 for the maze route. Note that this formulation directly accounts for both bends (vias) and wirelength. L-shapes are the most preferred route, as they have the fewest number of bends – zero or one. After Ls, Z-shapes are the most preferred, as they have the same (minimal) wirelength and only one extra bend. Next, C-shapes have an additional two units of wirelength and one additional bend. Finally, a maze route used as the last choice. In practice, the maze routes have more bends and wirelength than any of the other patterns. The chosen coeffients both encourage the use of short (L-shapes) routes as well as enable a degree of flexibility for detours. All experiments were performed on an AMD Opteron 2.4 GHz machine with 4 GB of memory.

Routability results for Sidewinder on the ISPD98 benchmarks [10] are shown in Table 1. We list the percentage of nets routed by Sidewinder, the number of iterations necessary and the total runtime for each benchmark. The ILP portion of Sidewinder is successful in routing 99.86% of all nets. Note that 100% routability is not required - the percentage of unrouted nets after ILP are trivial and a detail router is able to compensate (see Fig. 2). In order to compare directly with BoxRouter 1.0 and FGR 1.0, we take the solutions generated by Sidewinder and route all unrouted nets with a *single pass* of a maze router. No nets originally

[1]Except in cases where the L is degenerate — a flat wire



(a) (b) (c)

Figure 7: Via count comparison between Sidewinder and BoxRouter 1.0 for (a) ibm03, (b) ibm07, and (c) ibm10. The x- and y-axes have the number of vias for Sidewinder and BoxRouter 1.0, respectively. Each net is represented by a point whose coordinates are the number of vias it has in the results of these two routers. The blue line shows where Sidewinder and BoxRouter 1.0 use the same number of vias for a given net. Thus, if a point is above the blue line, Sidewinder uses fewer vias than BoxRouter 1.0 for the same net.

ISPD98 Bench -mark	Sidewinder			BoxRouter 1.0			FGR 1.0		
	Over- flow	Via count	Routed length	Over- flow	Via count	Routed length	Over- flow	Via count	Routed length
ibm01	255	15084	66058	102	15434	65588	0	17124	63332
ibm02	8	30668	174062	33	32529	178759	0	37937	168918
ibm03	0	22809	147524	0	25724	151299	0	31993	146412
ibm04	618	28611	172652	309	30836	173289	0	38464	167101
ibm05	0	50321	409778	0	51228	409747	0	77104	409739
ibm06	0	42847	280007	0	45692	282325	0	57036	277608
ibm07	0	56895	381694	53	60832	378876	0	78563	366180
ibm08	0	69321	413300	0	75291	415025	0	93905	404714
ibm09	0	64419	416554	0	68707	418615	0	86645	413053
ibm10	0	95316	591036	0	100546	593186	0	128141	578795
Average					+6.4%	+0.5%		+35.8%	-1.9%

Table 2: Solution quality comparison of Sidewinder to BoxRouter 1.0 [3] and FGR 1.0 [17]. Note that on these benchmarks, unlike the ISPD 2007 benchmarks, the default mode of FGR 1.0 does not penalize bends and only minimizes wirelength without accounting for vias.

routed by Sidewinder were ripped-up to route the remaining nets.

Table 2 compares these fully routed solutions to those of FGR 1.0 and BoxRouter 1.0 in terms of total overflow, via count and total routed wirelength. We first compare against FGR 1.0 [17], which won the ISPD 2007 Contest [11] in the 2D Category. While FGR 1.0 completes all the ISPD98 benchmarks without violation, its via counts are higher than Sidewinder's by 35.8%. Note that since this set of benchmarks don't formally have vias, we refer to vias as when a net "bends". That is, a via is counted when a horizontal routing segment is followed by a vertical segment (or vice versa). FGR 1.0, in this case, did not penalize bends and only minimized wirelength.

Compared against BoxRouter 1.0, we achieve 6.4% fewer vias and 0.5% shorter routed wirelength with moderate amounts of overflow. The via comparison is further depicted in Figure 7. The blue line represents where both routers use the same number of vias for that net. That is, a data point above the blue line means Sidewinder uses fewer vias and a data point below the blue line indicates Sidewinder uses more vias. Against BoxRouter 1.0, Sidewinder uses fewer vias on the vast majority of the nets. Using more sophisticated techniques such as iterations of rip-up and re-route, we could improve these violation counts. However, Sidewinder's solutions are sufficient to be used by a detail router.

5. CONCLUSIONS

In this paper, we propose the first ILP router that can handle the entire global routing grid and produces routing solutions with very few vias. Our path selection algorithm is congestion-driven - during each iteration, the algorithm intelligently selects the two best (least congested) paths as candidates based on a dynamically updated congestion map. Our ILP formulation is scalable: for a net $n \in N$, we only consider two possibilities. Thus, given $|N|$ nets, we only need $2|N|$ variables. In addition to the traditional L and Z routing patterns, we introduce shapes with detouring, C shapes, to significantly improve routability. Finally, our formulation guarantees that each new solution will be no worse than the current solution. Note that our incremental ILP approach is not limited to global ASIC routing - it is adaptable to detail routing applications as well as FPGA routing.

Our ILP formulation can be easily extended to various aspects of detail routing. In particular, mutual exclusion constraints and logical implications can be expressed compactly. This type of framework allows designers to easily handle complex design rules. One important application is forbidden pitches - routed paths must comply with multiple distance bounds with respect to either. For example, two paths can be be $[2\lambda, 4\lambda]$ or $[8\lambda, 10\lambda]$ apart[2]; $(4\lambda, 8\lambda)$ is strictly forbidden. Typically, complying with these restrictions is incredibly difficult, as there is no efficient way of modeling these limitations. However, using ILP, not only can these design constraints be easily expressed, they can also be integrated with other design rules and models (and solved optimally). Assignment of routing tracks in detailed routing can also be expressed efficiently using ILP formulations similar

[2] λ is a technology-dependent distance metric

to ours. As demonstrated in Section 4, Sidewinder is adept at handling pricing for vias.

6. REFERENCES

[1] C. Albrecht, "Global Routing by New Approximations for Multicommodity Flow," *IEEE TCAD* 20(5), pp. 622-632, 2001.

[2] M. Burstein and R. Pelavin, "Hierarchical Wire Routing,", *IEEE TCAD* 2(4), pp. 223-234, 1983.

[3] M. Cho and D. Z. Pan, "BoxRouter: A New Global Router Based on Box Expansion and Progressive ILP," In Proc. *DAC*, pp. 373-378, 2006.

[4] M. Cho, H. Xiang, R. Puri and D. Z. Pan, "Wire Density Driven Global Routing for CMP Variation and Timing," In Proc. *ICCAD*, pp. 487-492, 2006.

[5] C. C. N. Chu and Y.-C. Wong, "Fast and Accurate Rectilinear Steiner Minimal Tree Algorithm for VLSI Design," In Proc. *ISPD*, pp. 28-35, 2005.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. Section 24.3: Dijkstra's algorithm, pp. 595-601.

[7] R. Hadsell and P. H. Madden, "Improved Global Routing through Congestion Estimation," In *DAC*, pp. 28-34, 2003.

[8] J. Hu and S. S. Sapatnekar, "A Survey on Multi-net Global Routing for Integrated Circuits," *Integration, the VLSI Journal* 31(1), pp. 1-49, 2001.

[9] ILOG CPLEX: High-performance software for mathematical programming and optimization. `http://www.ilog.com/products/cplex`

[10] ISPD 1998 Global Routing benchmark suite. `http://www.ece.ucsb.edu/~kastner/labyrinth`

[11] ISPD 2007 Global Routing Contest and benchmark suite. `http://www.ispd.cc/ispd07_contest.html`

[12] R. Karp, "Complexity of computer computations", In *Reducibility Among Combinatorial Problems*, New York: Plenum, 1972.

[13] R. Kastner, E. Bozorgzadeh and M. Sarrafzadeh, "Pattern Routing: Use and Theory for Increasing Predictability and Avoiding Coupling," *IEEE TCAD* 21(7), pp. 777-790, 2002.

[14] G. Nam, F. Aloul, K. Sakallah and R. Rutenbar, "A comparative study of two Boolean formulations of FPGA detailed routing constraints", In Proc. *ISPD*, pp. 222-227, 2001.

[15] M. Pan and C. Chu, "FastRoute: A Step to Integrate Global Routing into Placement," In Proc. *ICCAD*, pp. 464-471, 2006.

[16] M. Pan and C. Chu, "FastRoute 2.0: A High-quality and Efficient Global Router," In Proc. *ASP-DAC*, pp. 250-255, 2007.

[17] J. Roy and I. Markov, "High-performance Routing at the Nanometer Scale," In Proc. *ICCAD*, 2007.

[18] TSMC: Silicon Success `http://www.tsmc.com/download/enliterature/html-newsletter/September03/InDepth/index.html`.

[19] J. Westra, C. Bartels and P. Groeneveld, "Probabilistic Congestion Prediction," In Proc. *ISPD*, pp. 204-209, 2004.

[20] H. Xu, R. Rutenbar and K. Sakallah, "sub-SAT: A Formulation for Relaxed Boolean Satisfiability with Applications in Routing," In Proc. *ISPD*, pp. 182-187, 2002.