Ordered Escape Routing Based on Boolean Satisfiability

Lijuan Luo and Martin D.F. Wong
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
{lluo3,mdfwong}@uiuc.edu

Abstract— **Routing for high-speed boards is largely a time-consuming manual task today. In this work we consider the ordered escape routing problem which is a key problem in board-level routing. All existing approaches to this problem cannot guarantee to find a routing solution even if one exists. We present an algorithm to exactly solve this problem based on Boolean satisfiability. Experimental results on escape routing problems from industry show that our algorithm performs well.**

I. Introduction

With ever increasing pin counts on denser boards, existing CAD tools fail to provide an automatic solution to board-level routing. As a result, routing for high-speed boards is largely a time-consuming manual task today. In this paper, we consider the ordered escape routing problem which is a key problem in board-level routing.

The input to the ordered escape routing problem is a pin array component and a required ordering of escaped pins on the component boundary. The required pin ordering is a way to ensure routing between components can be done in a planar fashion. Without loss of generality, we may assume that the escape pin ordering is 1, 2, 3, $\cdots$, n where n is the number of pins, by suitably renaming the pins if needed. In the escape problem, we are asked to find disjoint paths to route all pins within the pin array to the component boundary while satisfying the escape pin ordering. Fig. 1(a) shows an example of the ordered escape problem where all pins are required to escape at the right side of the component. This is a 1-side escape problem. We can also specify escapes to be on 2 sides, 3 sides, or 4 sides. Fig. 1(b) shows a 4-side escape problem. (Although 4-side escape is permissible, only 2 sides are actually used in this solution)

There is a large body of literatures [1][2][3][4] on the "un-ordered" escape routing problem, i.e., there is no constraint on the ordering of the escape pins. It is well known that the un-ordered problem can be solved optimally by a network flow approach. However, when it comes to ordered escape, all existing approaches [5][6] to the problem cannot guarantee to find a routing solution even if one exists. (Actually, [5] and [6] only solve a more restricted problem, where each pin has a fixed escape destination. However, we believe their approach can be extended to allow floating escape destinations, but the escape side for each pin has to be fixed.) The algorithm in [5] can only route nets in monotonic (or detour-free) patterns. Note that the solution in Fig. 1(a) is non-monotonic (Net 2's route is a detour) and the problem can not be solved by monotonic routing. The most recent work [6], although partially uses Integer Linear Programming, heuristically considers non-monotonic routing only when the routing graph has a cycle. Note that [6] can't find the routing solution in Fig. 1(a), where detour is needed for routing resource conflict, even though no cycle exists in the routing graph. Also, the 4-side escape problem in Fig. 1(b) can't be solved by both [5] and [6], since there is no way to know the escape side of each pin in advance.

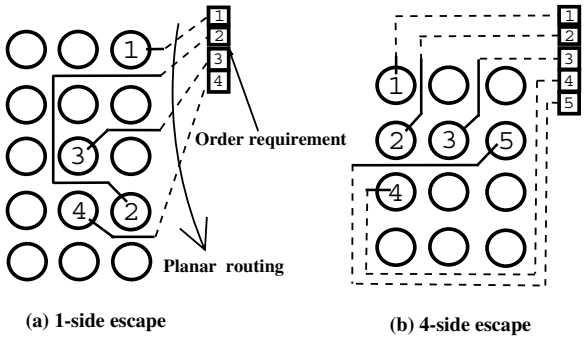

Fig. 1. Ordered escape routing problem

We present in this paper an algorithm to exactly solve the escape routing problem based on Boolean satisfiability. Experimental results on escape problems from industry show that our algorithm performs well.

II. SAT Formulation

A SAT problem is to find an assignment of variables to staisfy the equation

$$F(x_1, x_2, ..., x_n) = 1 \tag{1}$$

where F is a Boolean function with variables $x_1, x_2, \cdots, x_n$. Most often, F is expressed in Conjunctive Normal Form (CNF). A CNF expression is a logical **and** of one or more clauses, where each clause is a logical **or** of one or more literals. A literal is either the positive or the negative form of a variable.

Although the SAT approach has been widely used in CAD applications such as testing and verification [10][11][12][13][14], its application to physical design problems has been very limited [7][8]. This is because of the

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

myth that Boolean SAT instances in physical layout problems are always too large to be computationally feasible [9]. However, significant progress has been made in SAT solvers such that state-of-the-art SAT solvers can solve CNF formulas of millions of variables and clauses [15]. Moreover, we observe that most of the escape routing problem instances extracted from industrial boards are of moderate sizes, making SAT potentially a viable approach. In this paper, we provide a practical approach to optimally solve the escape routing problem using a SAT formulation.

A. Construct Boolean functions

For a given pin array, we construct its corresponding routing grid G. (See Fig. 2, which is the routing grid for Fig. 1(b).) We place additional cells called slots adjacent to the boundary of the routing grid to model the locations of the escape routes exiting the pin array. We define a graph on this routing grid. There are three types of nodes. Each free grid cell is called a grid node, each pin is called a pin node and each slot is called a slot node. An edge (shown as a dashed line in Fig. 2) is a connection between two neighboring nodes. We note that, due to the physical nature of the pin array, each pin can only connect "diagonally" to four neighboring grid nodes in the NW, NE, SW, and SE directions, and each pin-slot connection can only be horizontal or vertical.

To formulate the escape routing problem as a SAT problem, we define three types of Boolean variables: grid variable, slot variable and edge variable. For each grid node in G, we introduce a grid variable which is a string of Boolean variables corresponding to the binary representation of the index of the net routing through the node. Note that we can introduce a pin variable for each pin in a similar way except the string of Boolean variables is a constant (equals to the net index of the pin). Similarly, for each slot node, we introduce a slot variable, which is a string of Boolean variables corresponding to the binary representation of the index of the net exiting the pin array at that slot. Finally, an edge variable is a Boolean variable defined for each edge where it is 1 if and only if a net is routed on the edge.

We now show that the escape routing problem can be transformed into a SAT problem of the form

$$F = F_G F_E F_P F_S = 1 \tag{2}$$

where G, E, P, and S are the sets of all grid nodes, edges, pin nodes, and slot nodes, respectively. In other words, we are looking for a variable assignment to simultaneously satisfy the equations $F_G = 1$, $F_E = 1$, $F_P = 1$, and $F_S = 1$. Essentially, these equations specify various feasibility conditions on the Boolean variables in a feasible escape routing solution. The details of the constructions of F_G, F_E, F_P, and F_S are given in the following.

1) Construct F_G: For each grid node g, there are at most 8 edges attached to it, corresponding to the connections with 4 neighboring grid nodes and 4 neighboring pin nodes. It is obvious that the number of attached edges can't be fewer than 2. The variable assignment must satisfy the following

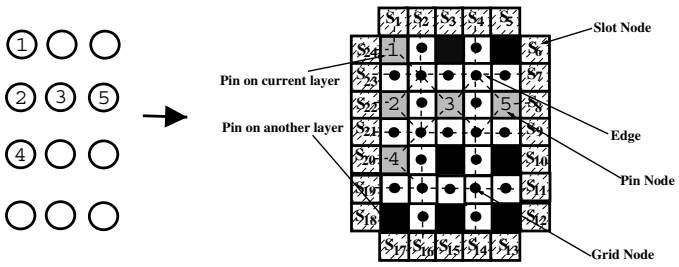

Fig. 2. Variable definition and slot index

condition: Among all the edges attached to g, either no edge variable is equal to 1 or there are exactly two edge variables equal to 1. In other words, either g is not used in routing, or it is used by routing one net from one neighboring edge to another neighboring edge. Let $e_1, \cdots, e_m$ be all the edges attached to node g, $(2 \leq m \leq 8)$. We define a Boolean function f_g for this condition.

$$f_g = (\prod_{i=1}^{m} \overline{e}_i) + (\sum_{\substack{i,j=1 \\ i \neq j}}^{m} (e_i e_j (\prod_{\substack{k=1 \\ k \neq i,j}}^{m} \overline{e}_k))) \tag{3}$$

For example, the grid node highlighted in Fig.2 has four neighboring edges e_1, e_2, e_3, e_4. For this node, we have

$$\begin{aligned} f_g \quad = \quad & (\overline{e}_1\overline{e}_2\overline{e}_3\overline{e}_3) + (e_1e_2\overline{e}_3\overline{e}_4) + (e_1\overline{e}_2e_3\overline{e}_4) + (e_1\overline{e}_2\overline{e}_3e_4) \\ & + (\overline{e}_1e_2e_3\overline{e}_4) + (\overline{e}_1e_2\overline{e}_3e_4) + (\overline{e}_1\overline{e}_2e_3e_4) \end{aligned} \tag{4}$$

Note that in each clause, either no edge variable or exactly two variables are positive.

Letting f_g range over all grid nodes, we get

$$F_G = \prod_{g \in G} f_g \tag{5}$$

2) Construct F_E: For each edge e, either the edge variable is equal to 0, or the two nodes connected by e are equal. This is obvious, since the edge can not be used simultaneously by two nets. Suppose the two node variables connected to e are n_1 and n_2. Define f_e to be the function corresponding to the above requirement. In the following, we also use e to represent the edge variable. Then we have

$$f_e = \overline{e} + (n_1 = n_2) \tag{6}$$

Letting f_e range over all edges, we get

$$F_E = \prod_{e \in E} f_e \tag{7}$$

3) Construct F_P: For each pin p there should be one and only one neighboring edge equal to 1, corresponding to the edge through which the net routes away from p. Let $e_1, \cdots, e_m$ be all the edges attached to p, We define f_p as follows:

$$f_p = \sum_{i=1}^{m} (e_i (\prod_{\substack{j=1 \\ j \neq i}}^{m} \overline{e}_j)) \tag{8}$$

For example, consider Pin 4 in Fig. 2, which has three neighboring edges e_1, e_2, e_3, then we have

$$f_p = (e_1\overline{e}_2\overline{e}_3) + (\overline{e}_1e_2\overline{e}_3) + (\overline{e}_1\overline{e}_2e_3) \tag{9}$$

Letting f_p range over all pins, we get

$$F_P = \prod_{p\in P} f_p \tag{10}$$

4) Construct F_S: This is for satisfying the given escape ordering. We first index the slots in clockwise order starting from the left topmost slot as shown in Fig. 2. (The starting point can also be changed to any other slot.) Use S_i to stand for the ith slot, and s_i for the slot variable. Then an escape ordering is correct if and only if for each pair i, j, where $i < j$ and S_i, S_j are both involved in routing, we have $s_i < s_j$. However, since we can't predict which slots are used in routing in advance, it is hard to come up with a Boolean function to define this order condition. Thus we propose another order condition: An escape order is correct if and only if for each pair i, j, where $i < j$, we have $s_i \leq s_j$. To prove the correctness of the second condition, we compare its differences from the previous one. First, for the two slots used by routing, we require $s_i \leq s_j$ instead of $s_i < s_j$. The two requirements are actually equivalent, since it is not possible for the two slots used by the same net, and $s_i = s_j$ is never true. Second, empty slots (slots not used in routing) are involved in the second condition. This requirement is equivalent to the first condition. Since the variable of an empty slot does not have any real meaning, we could give it any value. Suppose we get a value assignment which satisfies the first condition, then we can always make this assignment satisfy the second condition by setting the values of the empty slot variables in the following way. If s_i is an empty slot variable, then let $s_i = s_j$, where $j < i$ and S_j is the closest non-empty slot. If S_j does not exist, set $s_i = 0$. Hence, the second condition is totally equivalent to the first one. And now we can define F_S as follows:

$$F_S = \prod_{i=2}^{|S|}(s_{i-1} \leq s_i) \tag{11}$$

B. Transform into CNF

Nearly all the Boolean functions introduced in Section II.A are not expressed in CNF. In order to transform them into concise CNF expressions, we use several techniques, which will be introduced in the following.

1) Negative expression and De Morgan's law

Instead of constructing the Boolean function by enumerating all the legal cases as before, we enumerate all the illegal ones and write a negative expression. For example, for the f_p in (9), we have

$$\begin{aligned}&(e_1\overline{e}_2\overline{e}_3) + (\overline{e}_1e_2\overline{e}_3) + (\overline{e}_1\overline{e}_2e_3)\\ &\Leftrightarrow \overline{(\overline{e}_1\overline{e}_2\overline{e}_3) + (\overline{e}_1e_2e_3) + (e_1\overline{e}_2e_3) + (e_1e_2\overline{e}_3) + (e_1e_2e_3)}\\ &\Leftrightarrow (e_1 + e_2 + e_3)(e_1 + \overline{e}_2 + \overline{e}_3)(\overline{e}_1 + e_2 + \overline{e}_3)\\ &\quad (\overline{e}_1 + \overline{e}_2 + e_3)(\overline{e}_1 + \overline{e}_2 + \overline{e}_3)\ (De\ Morgan's\ law)\end{aligned} \tag{12}$$

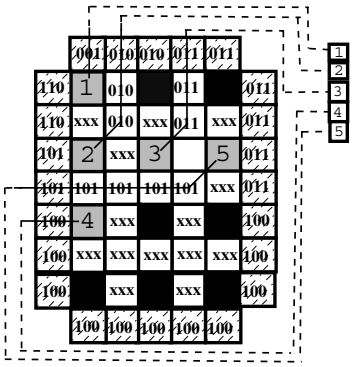

Fig. 3. SAT solution

2) Recursive procedure

Some Boolean functions constructed in Section II.A are based on strings of Boolean variables, such as $a = b$, $a \leq b$, where a and b are two strings of Boolean variables. For convenience of explanation, we suppose both of them have only two bits, i.e. $a = a_1a_2$, $b = b_1b_2$. The method for transforming $a = b$ to CNF is quite straightforward,

$$\begin{aligned}a = b &\Leftrightarrow (a_1 = b_1)(a_2 = b_2)\\ &\Leftrightarrow (a_1 + \overline{b}_1)(\overline{a}_1 + b_1)(a_2 + \overline{b}_2)(\overline{a}_2 + b_2)\end{aligned} \tag{13}$$

However, the transformation for $a \leq b$ is much more complex. It not only needs using a negative expression and De Morgan's law, but also a recursive procedure. First, we have

$$a \leq b \Leftrightarrow \overline{a > b} \tag{14}$$

We can enumerate all the possible cases for $a > b$. There are only three cases: (1) $a =$ 1x, $b =$ 0x, (2) $a = 11$, $b =$x0, (3) $a =$x1, $b = 00$, where "x" means "don't care". (Note that the three cases are not exclusive with each other.) So we have

$$\begin{aligned}a \leq b &\Leftrightarrow \overline{a > b}\\ &\Leftrightarrow \overline{(a_1\overline{b}_1) + (a_1a_2\overline{b}_2) + (\overline{b}_1a_2\overline{b}_2)}\\ &\Leftrightarrow (\overline{a}_1 + b_1)(\overline{a}_1 + \overline{a}_2 + b_2)(b_1 + \overline{a}_2 + b_2)\end{aligned} \tag{15}$$

For a general case, where $a = a_1 \ldots a_k$, $b = b_1 \ldots b_k$, we come up with a recursive procedure to enumerate the three possible cases for $a > b$. (1) $a_1 = 1$, $b_1 = 0$ (2) $a_1 = 1$, $a_2 \ldots a_k > b_2 \ldots b_k$, (3) $b_1 = 0$, $a_2 \ldots a_k > b_2 \ldots b_k$. Based on this recursive procedure, we can get the Boolean function for $a > b$, and then the Boolean function for $a \leq b$.

One solution for the problem in Fig. 2 is shown in Fig.3. Here, the number in each slot or free grid cell is the Boolean value of the slot variable or the node variable. For conciseness, only the edges, with the edge variables equal to 1 are drawn in the figure, the other edges are omitted.

III. Handling infeasible problems

Usually there is only one track used in each grid cell, since one track is the best for signal integrity. However, in some occasions, when no escape solution exists by using only one

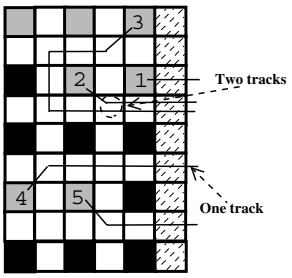

Fig. 4. 2-track solution

track, the design rule also tolerates another track in local regions, as shown in Fig. 4 . The SAT solution not only judges whether a one-track solution exists, but also gives a clue to add additional tracks when necessary. This judgement and dynamic track addition ability was never considered by the monotononic router [5] or the ILP method [6]. Moreover, in the field of SAT-based physical design algorithms, this is also the first work to take advantage of the unsatisfiable result.

A typical SAT solver works in this way: by continuously trying variable assignments and learning from the failed assignments, the solver finds more and more necessary variable values for satisfying the boolean function. There are two ending conditions for this search process: one is successfully finding a set of variable assignments, which make the function equal to 1; the other is finding a conflict clause, which can never be true based on the necessary variable values. In the second case, the conflict clause gives us hints about the cause of the failed escape, or the regions where additional tracks are needed.

Define the variables in the conflict clause to be conflict variables. Based on the types of conflict variables, we have different track addition strategies.

Hint 1 A node variable is a conflict variable. In this case, the most possible reason for the failure of routing is that several nets are competing for this node. Once this node is allocated to one net, the routing of the other nets can not be finished and conflict happens. The solution is to add an additional track within this node. For Fig. 4, the variable corresponding to the highlighted node is a conflict variable.

Hint 2 An edge variable, which is attached to a pin, is a conflict variable. In this case, the possible reason for the failure of routing is that the net is routing away from the pin in a wrong direction. If the variable has a positive sign in the conflict clause, then the correct direction to route the net away from the pin should be along this edge. This is obvious, since using this edge means the edge variable is equal to 1, and then the conflict clause is equal to 1. On the contrary, if the variable has a negative sign, then the correct direction should be any direction but the one along that edge. The router (actually the SAT solver, since we have transformed the routing problem into a SAT problem) failed to use the correct direction in the first place, because there was not enough routing resource there. Hence, the solution to this kind of conflict is adding tracks around the pin in the correct direction.

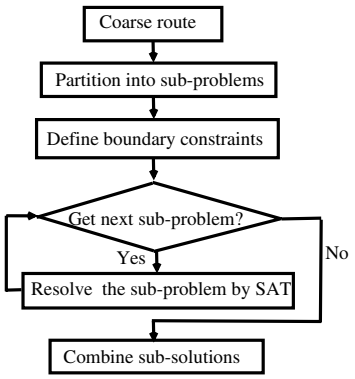

Fig. 5. Flow of SAT-involved routing engine

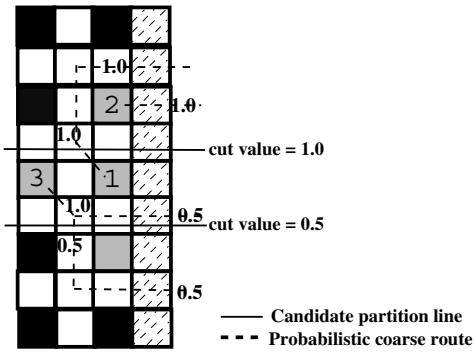

Fig. 6. Coarse route

Note that usually a conflict clause has more than one variables, so it is possible that Hint 1 and Hint 2 will be used together to resolve conflicts.

Hint 3 A slot variable is a conflict variable. In most of the cases, this kind of conflict should be handled in the same way as Hint 1. However, if the conflict clause is composed of only slot variables, this is a hint of ordering problem. This situation is more complex than the other two, since ordering problem is a global problem, which can not be easily solved by adding additional routing resource around the slots. Most often, a wrong ordering in slots is caused by choosing wrong directions when the nets route away from the pins. Thus the solution is to trace back to the pins of the conflict nets and add more routing tracks around these pins. Hopefully with the additional tracks, the router will find a way to route the nets away from the pins in correct directions.

IV. Solving large problems

The SAT approach introduced in Section II can also be embedded into other routing engines to resolve very large escape problems. This section introduces one possible engine. It works in the way as shown in Fig. 5. Several main modules are introduced as follows. **Coarse Router** An example of coarse routing is shown in Fig. 6(a). The coarse router routes all the boundary pins directly out of the grid without inward detour. (This is also the typical practice in industry.) The only exception occurs when direct escape violates order requirement, such as Net 1. All the remaining nets are routed

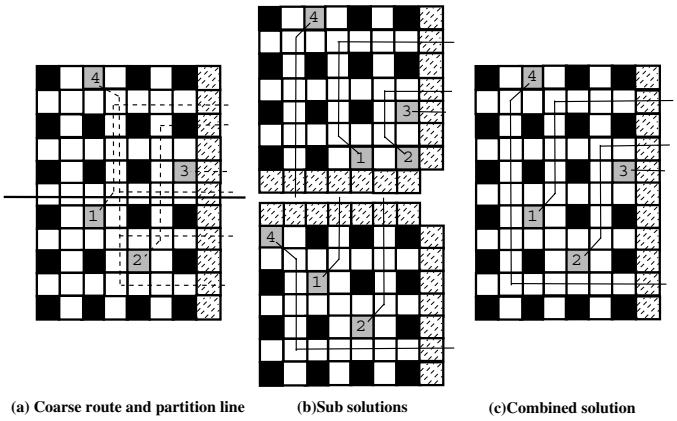

Fig. 7. Sub-problem definition and final solution

by L-shape pattern, and slots are evenly allocated to each net under the constraint of ordering requirement. If more than one slot belongs to a net, e.g. Net 3, then these slots are used with equal probability. The decimal on each routing segment is the probability value.

Partitioner Given a number U, we divide the original grid into sub-grids, each with at most U rows. (For simplicity, only one-dimensional partition is discussed here.) Under this constraint, we try to minimize the number of nets cut by sub-grid boundaries, since we want to limit the routing within each sub-grid. Note that the cut number is usually a decimal since we use the probabilistic course routing. Take Fig. 7 for example. Suppose $U = 3$, then there are only two candidate cuts which satisfy the sub-problem size constraint. We choose the lower cut with cut value equal to 0.5, and then the routing of all the three nets will be limited in one sub-grid.

Boundary Constraints If there are global nets (nets routing through several sub-grids) after partition, this module is used to define additional constraints for global nets. Take Fig. 7(a) for example. There are three global nets. Net 1 and Net 2 need to pass from the lower sub-grid to the upper one, while Net 4 from the upper to the lower. To guarantee sub-solutions will be finally combined into a complete solution in planar fashion, we try two possible orderings on the boundary: the nets routing downward being to the left of the nets routing upward, or vice versa. For the example problem, when we try the first choice, i.e. Net 4 is to the left side of Net 1 and Net 2 on the boundary, the two sub-problems are defined as in Fig. 7(b). Take the definition for the lower sub-problem for example. First, add another row of pins at the top of this sub-grid, and pin 4 is placed on the left end of this row, since boundary constraint requires it to be left. Second, to make sure Net 1 and Net 2 will route into the upper sub-grid,we require that Net 1 and Net 2 should escape only through the upper boundary. Fig. 7(c) shows the final routing result.

V. Experimental Results

We implemented the SAT based escape routing algorithm in C++. The whole SAT based router includes three modules. One is the pure SAT router introduced in Section II. If the escape pattern does not exist with current routing resource, the second module adds additional tracks in resource-insufficient area and then calls the first module again. This process iterates until a solution is found, or any more tracks are unacceptable. For large cases, which pure SAT router can't handle, the third module is used. It exploits the partition technique in Section IV, and calls the first module, and also the second module if needed, for each sub-grid.

We performed the experiments on a Pentium 4 2.8 GHz system with 4GB memory, and a Unix operating system. We use MiniSat [16] as the SAT solver. We tested four escape problems of real board design from industry, and two artificial cases (Case 3 and Case 6), which have even more complex escape patterns than industrial ones. The characteristics of these cases, as well as the experimental results, are shown in Table I. Each case has a #Row×#Col pin array as input.

TABLE I
Experimental Results

	#Row	#Col	#Pin	#Slot	#Var	#Clause	Run time
Case 1	13	4	16	25(1-side)	288	1782	0.02s
Case 2	11	5	25	39(3-side)	420	3369	0.94s
Case 3	10	6	25	41(3-side)	475	3509	0.14s
Case 4	10	10	15	76(4-side)	684	3214	6.93s
Case 5	17	6	32	44(2-side)	-	-	4.21s
Case 6	32	8	41	93(3-side)	-	-	1.85s

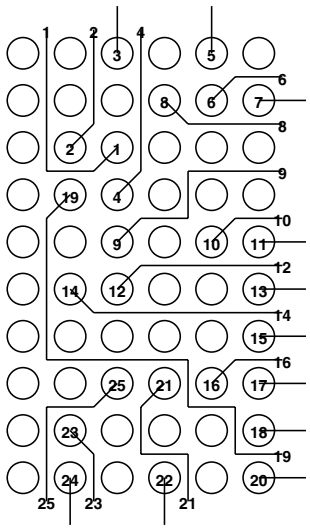

Fig. 8. Escape result for Case 3

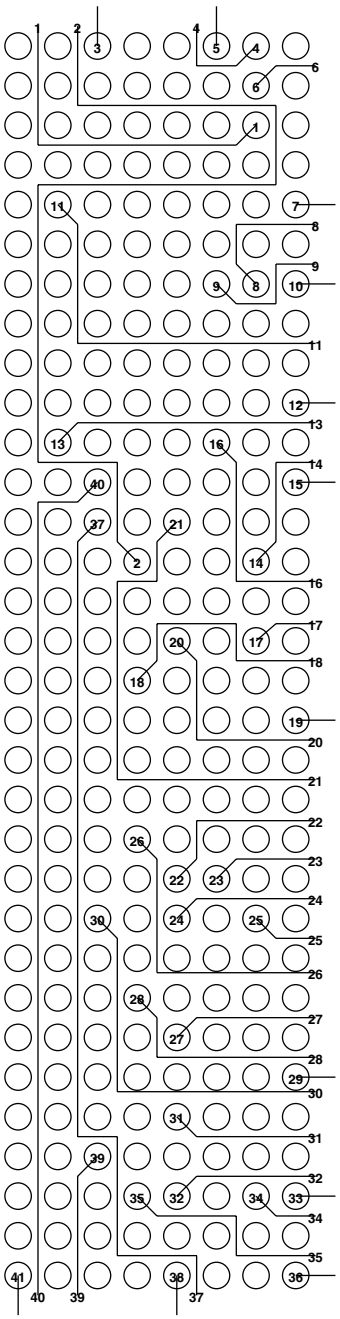

Fig. 9. Escape result for Case 6

#Pin is the number of pin terminals which need to be escaped on current layer. These cases cover 1-side, 2-side, 3-side, and also 4-side escape situations. The number and directions of escape sides are given as input, based on the distribution of routing resource outside the pin array. #Var and #Clause are the numbers of variables and clauses in SAT formulation. Case 2 and Case 5 have no solution under one track constraint, so additional tracks were added based on the technique introduced in Section III. The final two cases have large scales and were resolved by using the partition technique of Section IV. Since the two cases are corresponding to several SAT problems after partition, we did not fill in the #Var and #Clause columns for them. The escape results for the artificial cases are shown in Fig. 8 and Fig. 9.

Extensive experiments show that for pure SAT router, the maximum scale it can handle in acceptable time (within ten minutes for us) is approximately 10×10 pin array and the number of nets to be escaped does not affect the run time as much as the size of the pin array. For further larger cases, partition technique is necessary. Fig. 9 shows the ability of the escape router for finding non-trivial escape patterns for a very large case.

VI. Conclusion

A SAT based ordered escape routing approach is proposed in this paper. For the moderate cases from industry, it guarantees to find the solutions in reasonable CPU time. It can also handle the infeasible cases by dynamically adding tracks, and large scale cases by partition technique.

VII. Acknowledgement

This work was partially supported by the National Science Foundation under grant CCF-0701821 and a grant from the Fujitsu Laboratories.

References

[1] V. P. Roychowdhury, J. Bruck, and T. Kailath, "Efficient Algorithms for Reconfiguration in VLSI/WSI Arrays", IEEE trans. on Comp., 39(4): 480-489, April 1990.

[2] Y. Birk, and J. B. Lotspiech, "On Finding Non-Intersecting Straightline Connections of Grid Points to the Boundary", J. of Algorithms, 13(4): 636-656, Dec. 1992.

[3] W.-T. Chan, and F. Y. L. Chin, "Efficient Algorithms for Finding Maximum Number of Disjoint Paths in Grids", SODA, pp. 454-463, 1997.

[4] J. W. Fang, I. J. Lin, P. H. Yuh, Y. W. Chang, and J. H. Wang, "A Routing Algorithm for Flip-Chip Design," Proc. of ICCAD, pp. 753-758, 2005.

[5] Y. Tomioka, and A. Takahashi, "Monotonic Parallel and Orthogonal Routing for Single-Layer Ball Grid Array Packages," Proc. of ASP-DAC, pp.642-647, 2006

[6] J.W. Fang, C.H. Hsu, and Y.W. Chang, "An Integeer Linear Programming Based Routing Algorithm for Flip-Chip Design", Proc. of DAC, pp. 606-611, 2007.

[7] S. Devadas, "Optimal Layout Via Boolean Satisfiability," Proc. of ACM/IEEE ICCAD, pp. 294-297, 1989.

[8] R. G. Wood, and R. A. Rutenbar, "FPGA Routing and Routability Estimation Via Boolean Satisfiability," IEEE Trans. VLSI Systems, pp. 222-231, June 1998.

[9] Gi-Joon Nam, "An Exploration of Physical Design Problems Via Boolean Satisfiability (SAT)", Ph.D forum at DAC, 1999

[10] T. Larrabee, "Eficient Generation of Test Patterns Using Boolean Satisfiabiliy", Ph.D. Dissertation, Department of Computer Science, Stanford University, STAN-CS-90-1302, February 1990.

[11] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability", IEEE Trans. on CAD, 11(1):4C15, Jan 1992.

[12] H. Konuk and T. Larrabee, "Explorations of Sequential ATPG Using Boolean Satisfiability", Proc. of the 11th IEEE VLSI Test Symposium, pp. 85-90, April 1993.

[13] P. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Combinational Test Generation Using Satisfiability," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 15, pp. 1167C1176, 1996.

[14] M. Velev, and R. Bryant, "Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors," Proc. of DAC, July 2001.

[15] E. Goldberg, "Practical SAT Solving: Achievements, Problems and Opportunities", Algorithms for the SAT-problem Workshop, 2006

[16] http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/