DpRouter: A Fast and Accurate Dynamic-Pattern-Based Global Routing Algorithm*

Zhen Cao[1], Tong Jing[1, 2], Jinjun Xiong[2, 3], Yu Hu[2], Lei He[2], Xianlong Hong[1]

[1] Computer Science & Technology Department
Tsinghua University
Beijing 100084, China
Phone: +86-10-62785564
e-mail: caoz@mails.thu.edu.cn

[2] Electrical Engineering Department
UCLA
Los Angeles, CA, 90095, USA
Phone: (310) 267-5407
e-mail: {tomjing, jinjun, hu, lhe}@ee.ucla.edu

[3] IBM Research Center
Yorktown Heights
NY, 10598, USA
Phone: 1-914-945-3676
e-mail: jinjun@us.ibm.com

Abstract - This paper presents a fast and accurate global routing algorithm, DpRouter, based on two efficient techniques: (1) dynamic pattern routing (Dpr), and (2) segment movement. These two techniques enable DpRouter to explore large solution space to achieve better routability with low time complexity. Compared with the state-of-the-arts, experimental results show that we consistently obtain better routing quality in terms of both congestion and wire length, while simultaneously achieving a more than 30x runtime speedup. We envision that this algorithm can be further leveraged in other routing applications, such as FPGA routing.

I Introduction

The success of future integrated circuit designs requires the consideration of physical impact. Congestion-aware global routing is crucial to achieve this goal and has been proven useful in physical synthesis.

Existing global routing algorithms mainly focus on congestion reduction [1-8]. Among them, [7] and [8] are two of the most recent publications with good results. Labyrinth [7] is a global router that evaluates routing results in terms of congestion and wire length. Fengshui [8] proposed the concept of amplified congestion estimation, which helped to improve both over-congestion and routing length.

Some Steiner tree algorithms [9-11] even tried to improve the routability. [9] and [10] presented the idea of solving Steiner tree problem based on pre-computed lookup table. [11] proposed the concept of flexibility in the rectilinear Steiner tree (RST) problem. These tree algorithms are helpful, but the final congestion reduction still mainly relies on powerful routing algorithms.

The problem of congestion-aware routing, however, is far-from being solved. This paper focuses on the routability-driven global routing problem. Our focus is congestion reduction and algorithm efficiency. The major contributions of this work are as follows:

(1) An efficient dynamical pattern routing (Dpr) technique to achieve the optimal routing solution for two-pin nets.

(2) A segment-move technique to extend the searching space for routability driven RST problem.

(3) A congestion-aware global routing algorithm that combines the above two techniques to achieve high speed and high optimization capability.

Compared with the state-of-the-art routers, labyrinth 1.1 [7] and Fengshui 5.1 router (newest edition of *Chi* dispersion router) [8], we simultaneously reduce total overflow by 89.76% and 51.71%, total wire length by 14.49% and 1%, while achieving 64x and 38x speed up of runtime, respectively. Comparison has also been performed between Dpr and Z-shaped pattern routing [6] on runtime and congestion reduction. We show that Dpr achieves better routing solutions with the same time complexity.

The rest of this paper is organized as follows. Basic definitions and the problem formulation are introduced in Section II. The dynamic pattern routing (Dpr) technique is described in Section III. In section IV, both DpRouter and the segment-move techniques are introduced in detail. Experimental results are given in Section V, and Section VI concludes this paper.

II. Preliminaries

A. Basic Definitions and Problem Formulation

Global routing problem is often formulated by partitioning the routing area into global cells and mapping all physical pins in each cell into the center of the cell. The centers make up the global routing graph (GRG). We introduce the following concepts in the context of routability driven global routing.

Given c_e as the capacity of a GRG edge, d_e as the routing demand for edge e, the overflow of edge e is defined as follows.

$$overflow_e = \begin{cases} d_e - c_e, & if\ d_e > c_e \\ 0, & otherwise \end{cases} \qquad (1)$$

The total overflow of the entire routing area is as follows.

$$tof = \sum_{e \in E} overflow_e \qquad (2)$$

Segment: a segment is a horizontal or vertical edge line on GRG connecting its two endpoints.

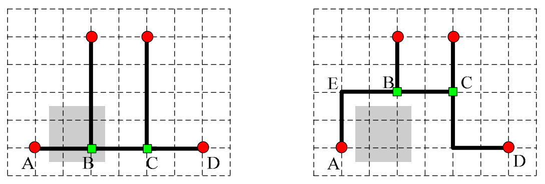
Edge of a tree: an edge of a tree is the routing between two vertexes in this tree. Note that the difference between edge of a tree and edge on GRG. As shown in Fig.1.(b), A and D are pins, B and C are Steiner points, the edge connecting AB consists of two segments, AE and EB.

Topology: for a given RST, there is a corresponding graph

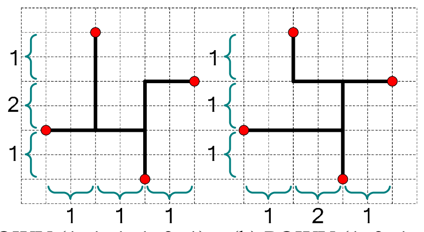
* This work was supported in part by the Key Project of Chinese Ministry of Education under Grant No.106008, the Specialized Research Fund for the Doctoral Program of Higher Education (SRFDP) of China under Grant No.20050003099, and the National Natural Science Foundation of China (NSFC) under Grant No. 90607001.

G(*T*,*E*) called topology, where *T* is the set of pins and Steiner points and *E* is the set of edges between the vertexes in *T*. For a given topology for a netlist, there may exist many embedded Manhattan trees.

[11] presented the definition of flexible edges. A flexible edge is an edge by which two vertexes are connected by more than one segment. Flexible edges allow more than one minimum length routing solution, which is more suitable for global routing than non-flexible edges. As shown in Fig.1, the tree in Fig.1(b) has two flexible edges (AB and CD), whereas the tree in Fig.1(a) has no flexible edges at all. So the tree in Fig.1(b) can avoid congested grey area by exploiting flexible edges to achieve the minimum length, whereas the tree in Fig.1(a) can not. Therefore we should exploit the flexibility of each tree as much as possible to gain better routing solutions.



(a) No flexible edge in tree (b) Two flexible edges in tree
Fig.1. RST with or without flexible edges
(gray area indicates congestion).



(a) POWV (1, 1, 1, 1, 2, 1) (b) POWV (1, 2, 1, 1, 1, 1)
Fig.2. Different POWVs for a given net

B. *Flute Technique*

[10] presented a fast and accurate rectilinear Steiner minimal tree (RSMT) algorithm based on lookup table, called FLUTE. In FLUTE, the set of all degree *n* nets are partitioned into *n!* categories according to their relative positions of pins. For every category, several potentially optimal wire length vectors (POWVs, i.e., the different linear combinations of distance between adjacent Hanan grid lines [12]) are stored in a lookup table to give a fast estimation of minimal wire length. As shown in Fig.2, two POWVs, (1, 1, 1, 1, 2, 1) and (1, 2, 1, 1, 1, 1), for one category of 4-pin net are stored in a look up table. Then, the minimal wire length can be estimated by calculating the wire length considering the real distance between adjacent Hanan grid lines. For high-degree nets, the table size will become impractical large. So a net-braking technique is used to divide the net into sub-nets recursively to find an approximated minimal wire length.

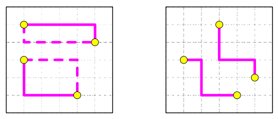
For every POWV, one tree topology is also stored in FLUTE. However, in global routing, only one tree topology is not enough. So we present a fast search technique to consider all possible routing solutions for every POWV, which will be introduced in Section III and Section IV.

III. Dynamic Pattern Routing

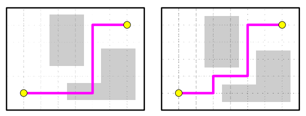
In global routing, researchers often decompose a tree topology into 2-pin nets. Then they use pattern routing or maze routing to route each 2-pin net separately. But the searching space of L- or Z-shape pattern routing is too small and the time complexity of maze routing is too high. Here we present a new and flexible technique to connect a 2-pin net, called dynamic pattern routing (Dpr). Compared with L- or Z-shape pattern routing, this technique can explore patterns with more than 2 bends and search the solution space of all routings with minimal wire length and low time complexity.

A. *Pattern Routing*

As shown in Fig.3, the L-shape (1-bend) pattern routing only allows routing on the bounding box, while Z-shape (2-bend) pattern routing only allows routing with two bends on or inside the bounding box. So compared with maze routing, the L- or Z-shape pattern routing's time complexity is much lower. However, in the routability driven routing problem, especially in the case of high congestion, L- or Z-shape pattern routing can not avoid some congested area, as shown in Fig.4(a). Therefore, new fast pattern routing technique with more-than-2-bend is needed for routability driven routing as shown in Fig.4(b).



(a) L-shape (b) Z-shape
Fig.3. L- and Z-shape pattern routing



(a) Z-shape (b) 3-bend
Fig.4. Pattern routing in congested area

B. *Dynamic Pattern Routing Algorithm*

We present an algorithm called dynamic pattern routing (Dpr) to search for more-than-2-bend routings with low time complexity.

Theorem 1: There are *C(m+n, m)* different routing solutions to connect an edge with minimal wire length, where *m* and *n* are the vertical and horizontal distances of the edge, respectively.

Proof: Every routing solution can be formulated as choosing *m* vertical steps out of total *m+n* steps. That is just *C(m+n, m)* different combinations ■

We employ a linear function to estimate the routing cost of every edge on GRG, while considering congestion and other information. Then routability driven routing problem is then formulated to find the routing solution with minimal cost. This function (formula (3)) is given in Sub-Section IV.E. However, it is too costly to consider all possible solutions. So we utilize dynamic programming in Dpr.

Theorem 2: With the linear cost function, the optimal routing solution must be optimal in its sub routings.

Proof: (By way of contradiction) Suppose that the routing with minimal cost has a sub-routing which is not optimal. Then, this sub-routing can be replaced with another routing with a lower cost. Since the cost function is linear, the so-obtained new routing would have a lower cost than before, which is contradictory to the assumption that the original solution is optimal. Therefore, all sub-routings of the optimal routing must be optimal too. ■

The key of dynamic programming is that the sub-solution of the optimal solution must be optimal too. Therefore, with Theorem 2, we can utilize dynamic programming to find the optimal routing of an edge. The idea is illustrated in Fig.5. To find an optimal routing from A to B, we only need to find the optimal routings form A to C and from A to D. Then, by adding the routing cost for CB and DB, we can choose the optimal routing from A to B out of the two candidates, which is ACB in the figure.

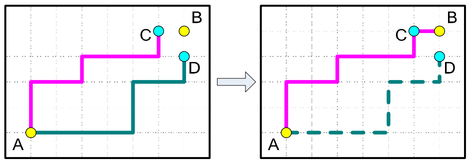
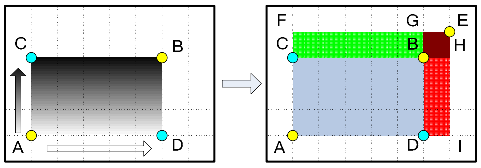


Fig.5. Dynamic pattern routing



(a) Dpr algorithm (b) Dpr-M algorithm
Fig.6. Dpr and Dpr-M algorithm

```
Algorithm 1 Dpr algorithm
Input: Routing graph $G$, a 2-pin net $n$={($x_1$, $y_1$), ($x_2$,$y_2$)}
Output: Optimal routing solution for $n$
1: if $x_1$ equals to $x_2$, or $y_1$ equals to $y_2$ then
2:     return the straight line routing from ($x_1$, $y_1$) to ($x_2$,$y_2$);
3: end if;
4: for $dy$ from 0 to $y_2$ - $y_1$ do
5:     for $dx$ from 0 to $x_2$ - $x_1$ do
6:       current position is ($x_1$+$dx$, $y_1$+$dy$), namely $p$;
7:       get two points and corresponding segments left and below $p$;
8:       calculate the corresponding cost for routings from
           ($x_1$, $y_1$) to $p$ that pass these two points;
9:       choose one with lower cost as $p$'s parent, record the cost;
10:    end for;
11: end for;
12: back trace from ($x_2$,$y_2$) to ($x_1$, $y_1$), get the routing solution $s$;
13: return $s$;
```

Fig.7. The pseudo-code of Dpr algorithm

With the above idea, we can find all optimal routing solutions from A to all vertexes on or inside the area of ACBD, shown in Fig.6.(a). The pseudo-code of Dpr is shown in Fig.7. Dpr first finds optimal routings for vertexes on line AD, then extend to the line just above AD. The iteration goes on until finally extend to CB.

Theorem 3: The time complexity of Dpr is the same as Z-shape pattern routing.

Proof: Given $n=\{(x_1,y_1),(x_2,y_2)\}$, let Z be the segments on and within the bounding box of n, L be the segments on the bounding box of n, then $|Z|= 2\times|x_1-x_2|\times|y_1-y_2|+|x_1-x_2|+|y_1-y_2|$, $|L|= 2\times(|x_1-x_2|+|y_1-y_2|)$. For L-shape pattern routing, time complexity is $O(|L|)$, while for Z-shape pattern routing it is $O(|Z|)$. With analysis of Dpr algorithm, we can find that each segment on or within the bounding box is only calculated once. So time complexity of Dpr is $O(|Z|)$, which is the same as Z-shape pattern routing. ■

Dpr is very powerful for routability driven global routing, because compared with Z-shape routing, with the same time complexity, the searching space of Dpr routing is much larger. Dpr obtains the optimal routing solution with minimal length by considering all possible $C(m+n,\ m)$ minimal wire-length routing solutions, while Z-shape only considers $m+n$ routing solutions. Our experimental results also confirm this analysis.

C. *Extension of Dpr*

Dpr technique is very flexible, and it enables us to solve the routing problem with movable endpoints (Dpr-M) efficiently. In this case, the endpoint of an edge is allowed to move in some directions. For example, as shown in Fig.6(b), we may first find optimal routings from A to all vertexes in rectangle ACBD. If the end point moves from B to E, we only need to extend the calculation to green (BCFG) and red areas (BHID). In another word, only the calculation to find the optimal routings for vertexes in the extended area is needed, hence saving much runtime.

In Section IV.B, we use a segment-move technique to explore such flexibility for better routability.

IV. Global Routing Algorithm DpRouter

A. *Global Routing Flow*

We present a global router based on the combination of Dpr and segment-move technique.

In our algorithm, the initial routing solution is found by Dpr and segment-move technique. After initial routing, first we rip-up and reroute by using Dpr and segment-move to reroute every congested net for several iterations. Then we use maze routing. The notion is that maze routing got a much higher time complexity and a larger searching space than Dpr and segment-move do. Therefore, we first use the quicker method to reduce congestion keeping short wire length. Then we use maze routing to reduce congestion of the nets that need to be detoured. This strategy can reduce run time while keeping good routability.

In our algorithm, we order the nets by size of the bounding box. In the initial routing step, the order is from small to large, while in the rip-up and rerouting the order is from large to small. The notion is that net with bigger bounding box is more flexible than net with a smaller one. So, in initial routing, big ones are routed after the small ones to avoid congestion. In rerouting, the information of congestion is enough. So, big nets should be rerouted firstly to quickly reduce congestion.

The flow of the routing algorithm DpRouter is shown in Fig.8. The detailed introduction of DpRouter will be given in the following.

```
Algorithm 2 DpRouter
Input: Routing graph G and nets set N
Output: Routing solutions for every net in N
1:  congestion estimation;
2:  for every net list l in N do
3:      get the POWVs set P of l;
4:      for every POWV v in P do
5:          use FLUTE to get the original topology t for vector v of
            net list l;
6:          find independent movable segment sets M of t;
7:          use Dpr and Segment-Move techniques to move segments
            in M to find the best routing solution s for v, put s in set S;
8:      end for;
9:      select the best solution s' in S, route it;
10: end for;
11: rip-up and reroute every net in congested area, using method the
    same of line 3-8, for several iterations;
12: rip-up and reroute every net in congested area, tree topology is se-
    lected using method of line 3-8, but maze routing is utilized in rout-
    ing;
13: return;
```

Fig.8. Pseudo-code of DpRouter.

B. *Segment-Move Technique*

As discussed in Sub-Section II.B, only one topology for each POWV is stored in FLUTE, which is not sufficient for global routing. Searching all solutions with minimal length for each POWV is needed by routability driven global routing. Then every different topology should be given.

This problem is shown in Fig.9, where the red square points are Steiner points, the dotted edge indicates this edge is a flexible one, the green edge (light color) indicates this edge is a movable segment and the gray area indicates congestion. Fig.9(a) is the original tree topology, stored by FLUTE. We can only search some of the solution space by giving different routing solutions for this topology, one of them is shown Fig.9(d). This is not enough for avoiding congestion. Note that some other tree topologies are transformable from the original one by moving the green segment up, as shown in Fig.9(b) and Fig.9(c). Then, a Dpr search of each edge in Fig.9(b) and Fig.9(c) can cover all the other solution space of this POWV, two of them are shown in Fig.9(e) and Fig.9(f). Finally, Fig.9(f) shows the best routing solution.

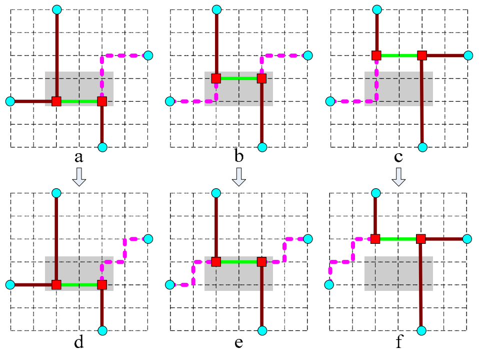


Fig.9. Different routing solutions for one POWV

We firstly try to find all the movable segments of the original tree topology and move them to new positions. Then the Dpr technique is utilized to search all edges to give the best routing for new topologies. But using Dpr to every topology separately is too costly. So, segment-move and Dpr should be combined to reduce the time complexity.

C. *Combination of Dpr and Segment-Move*

We first try to find all the movable segments of the original tree topology and move them to new positions. Then the Dpr technique is utilized to search all edges to give the best routing for new topologies. In another word, segment-move and Dpr are combined to further reduce the time complexity.

Three types of movable edges are shown in Fig.10.

Type A: the movement of the movable segment will not reduce the flexibility of its adjacent edges, which is shown in Fig.10(a) and Fig.10(b). In Fig.10(a) the movement of the segment will not influence the flexibility of other edges. In Fig.10(b), upwards movement will increase the flexibility of edge BA as shown in Fig.10(b').

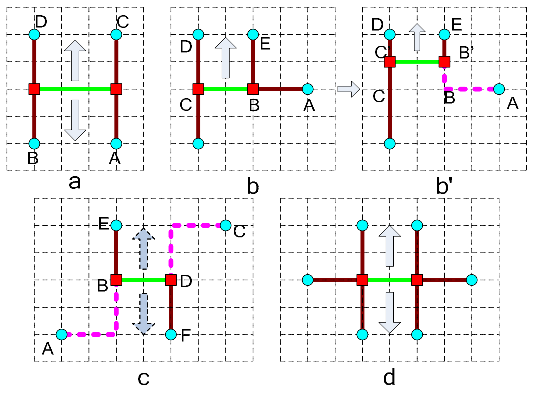


Fig.10. Different types of movable edges

Type B: movement of the movable segment will decrease flexibility of its adjacent edges, as shown in Fig.10(c). Upwards movement will decrease DC's flexibility and increase AB's.

Type C: movement of the movable segment will cause the change of connection relationship among the vertexes. As shown in Fig.10(d), downward or upward movement of the movable segment will increase the number of Steiner points and connection relationships of these vertexes.

For Type A, we first put the segment on one end of its movable range. In Fig.10(a), it can be the segment AB or CD. In Fig.10(b), it is the segment BC. Then the movable segment will move along its movable direction. We calculate the corresponding cost at each position. If the movement extends the flexibility of an edge, such as AB' in Fig.10(b'), then the Dpr-M algorithm will be used to find the changed optimal routing solution for this changed flexible edge.

For Type B, we use the similar method. If two flexible edges are impacted by the movement shown in Fig.10(c), we first use Dpr for each of the two flexible edges. Assume that its movable end point (i.e., the interconnect point) is on its maximum movable position, in this way we can get optimal routing of each possible position. When moving the segment, both edges' corresponding optimal routings are already known based on previous computation.

For Type C, we first change the tree topology. Then, the problem can be reclassified to either Type A or Type B.

D. *Overlap of Movable Segments*

We can move every movable segment with the combination of Dpr and segment-move. However, sometimes movable segments overlap so we cannot move them independently. Based on our analysis of all the relative positions of two movable segments, we summarize two types of overlap and show them in Fig.11.

Type A is shown in Fig.11(a). For this type, movement of one segment will not influence the flexible edges of the other. So movement is independent. But the tree topology may have to be changed as the point of B and C in Fig.11(b).

Type B is shown in Fig.11(c). For this type, the interconnected point A is the endpoint of both movable edges. Then, movement of these two movable segments is not independent. As shown in Fig.11(c), the maximum movable position for the interconnected point A is B. So A is movable in the gray area. We adopt Dpr-M in this case. When A is moving, both Steiner point C and D are moving too, shown in Fig.11(d). So incremental Dpr-M will be applied to these three flexible edges to find the optimal routing solution.

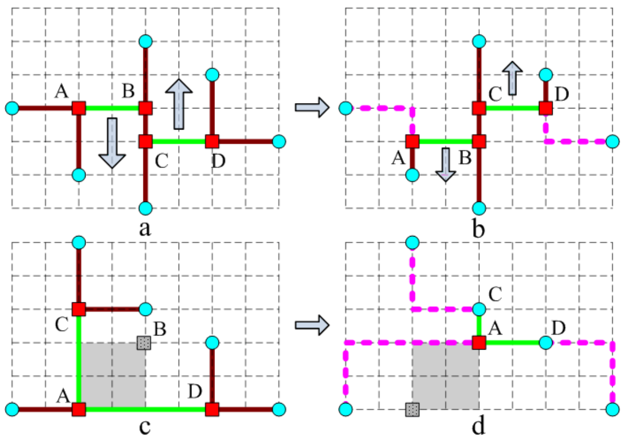


Fig.11. Two types of overlaps

E. *Selection of Tree Routing Solution*

A cost function is employed to select the best tree routing solution. We first find all movable segments and then divide them into independent movable sets. After that, we use Dpr-M to move every segment in independent movable set to minimize the total cost, which is defined as follows.

$$f = a * ov + b * mid + c * net + d * esti \tag{3}$$

where a, b, c, and d are coefficients, ov represents overflow, net and $esti$ represent real routed and estimated number of nets on these GRG edges, respectively, mid is the number of nets over lower capacity (To avoid congestion, we often give a lower capacity for each GRG edge, which is $T\%$ of its original capacity) of these GRG edges. The tree topology with the minimal cost is selected for the net list to be routed.

The coefficients were different in different steps of routing. Before maze routing, we give T as 50% and give a, b a small number. When performing maze routing, we raise T to 95% and raise a, b too. The idea is that, in maze routing, the weight of congested should be bigger to avoid congested area.

F. *Congestion estimation*

We use Dpr to do congestion estimation. To reduce runtime, we only choose the tree topology with minimal wire length from FLUTE and route it only using Dpr. After that, we record the routed net number on each GRG edge as the estimated congestion information.

The coefficient d in formula (3) is deceasing when more and more nets are routed, and finally down to zero in the rip-up reroute and maze routing steps. The reason is that in the last moment of routing, the routed nets will give more accurate congestion information than the estimated information.

V. Experimental Results

A. *Experiments Setup*

We implement our algorithm in C++, and we test our global router on ISPD'98 benchmarks. TABLE I shows the characteristics of all benchmarks.

We compare our router with two latest academic state-of-the-art ones, labyrinth 1.1 and Fengshui 5.1 (newest implementation of *Chi* dispersion router). The total overflow (*tof*), total wire length (*twl*) and runtime (CPU) of the routing results are compared and shown in TABLE II. Experiments in this paper are performed in 1.6GHz CPU Linux PC.

B. *Wire Length and Congestion Comparisons*

The comparison of test results is shown in TABLE II, from which we can find that DpRouter achieves high speed up with a great congestion reduction. Compared with Labyrinth 1.1, DpRouter reduces on average total overflow by 89.76%, total wire length by 14.49%, and a 64x speed up. Compared with Fengshui 5.1, DpRouter reduces total overflow by 51.71%, total wire length by 1%, and a 38x speed up.

C. *Congestion Reduction of Dpr*

We compared Dpr with L- or Z-shape pattern routing on *tof* and runtime. To demonstrate the optimization capability of Dpr, we choose the topology given by FLUTE with minimal wire length for all the 3 pattern routing techniques and give 5 iterations of rip-up and rerouting.

The results are shown in TABLE III. Dpr can reduce more than 30% overflow than L- or Z-shape pattern routing do with shorter runtime. The reason for longer runtime of Z-shape pattern routing is that it shares the same complexity with Dpr but has to route more congested nets than Dpr does in the rerouting step.

TABLE I
Benchmark Data

Circuits	Net#	Grids	Circuits	Net#	Grids
ibm01	13k	64×64	ibm06	34k	128×64
ibm02	19k	80×64	ibm07	46k	192×64
ibm03	26k	80×64	ibm08	49k	192×64
ibm04	31k	96×64	ibm09	59k	256×64
ibm05	30k	128×64	ibm10	66k	256×64

TABLE III
Comparison with L- and Z-shape Pattern Routing (PR)

circuit	L-shape PR		Z-shape PR		Dpr	
	tof	cpu(*s*)	*tof*	cpu(*s*)	*tof*	cpu(*s*)
ibm01	2118	0.13	2053	0.18	1636	0.15
ibm02	4081	0.37	4030	0.45	3585	0.37
ibm03	1016	0.27	983	0.42	480	0.27
ibm04	3163	0.32	3151	0.46	2356	0.31
ibm05	51	0.44	94	0.73	0	0.37
ibm06	3424	0.55	3083	0.82	2210	0.57
ibm07	2789	0.60	2644	0.84	2073	0.57
ibm08	2690	0.98	2580	1.45	1229	0.94
ibm09	4065	0.86	3528	1.62	1656	1.09
ibm10	3576	1.00	3306	1.84	2343	1.29
Total	26973	5.52	25452	8.81	17568	5.93
*Norm	1.54	0.93	1.45	1.49	1.00	1.00

* Normalized to Dpr's *tof* and CPU.

VI. Conclusions

This paper presents a very fast global routing algorithm. A dynamical pattern routing (Dpr) technique is presented to achieve optimal routing solutions for an edge with low time complexity. A segment-move technique is developed to search more solution space for routability driven RST problem. Based on the combination of Dpr and segment-move techniques, a global router called DpRouter is developed. We believe, though not verified, that this algorithm can be further leveraged in other routing applications, such as FPGA routing.

Acknowledgements

The authors would like to thank Prof. Chris Chu from Iowa State University for providing the source code of FLUTE.

References

[1] C. Chiang, M. Sarrafzadeh, and C.K. Wong, "Global routing based on steiner min-max trees", IEEE Trans. on CAD, 1990, 9(12): pp.1318-1325.

[2] W. Swartz and C. Sechen, "A new generalized row-basedglobal router", In Proc. of DAC, 1993, pp.491-498.

[3] R. C. Carden IV, J. M. Li, and C. K. Cheng, "A global router with a theoretical bound on the optimal solution", IEEE Trans. on CAD, 1996, 15(2): pp.208-216.

[4] C. Albrecht, "Provably good global routing by a new approximation algorithm for multicommodity flow", In Proc. of ISPD, 2000, pp. 19-25.

[5] T. Jing, X.L. Hong, H.Y. Bao, J.Y. Xu, and J. Gu, "SSTT: Efficient local search for GSI global routing", Journal of Compute Science and Technology, 2003, 18(5): pp.632-639.

[6] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Pattern routing: use and theory for increasing predictability and avoiding coupling," IEEE Trans. on CAD, 2002, 21(7): pp.777-790.

[7] Labyrinth, http://www.ece.ucsb.edu/~kastner/labyrinth/

[8] R.T. Hadsell and P.H. Madden, "Improved global routing through congestion estimation," In Proc. of DAC, 2003, pp.28-31.

[9] C. Chu, "FLUTE: fast lookup table based wirelength estimation technique", In Proc. of ICCAD, 2004, pp. 696-701.

[10] C. Chu and Y. Wong, "Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design," In Proc. of ISPD, 2005, pp. 28-35.

[11] E. Bozorgzadeh, R. Kastner, and M. Sarrafzadeh, "Creating and exploiting flexibility in rectilinear Steiner trees," IEEE Trans. on CAD, 2003, 22(5): pp.605-615.

[12] M. Hanan, "On Steiner's problem with rectilinear distance", SIAM J. on Applied Mathematics, 1966, 14: pp.255-265

TABLE II
Comparison between Labyrinth 1.1 and Fengshui 5.1.

circuit	Labyrinth 1.1			Fengshui 5.1			DpRouter			Improve on Labyrinth			Improve on Fengshui		
name	*twl*	*tof*	cpu(*s*)	*twl*	*tof*	cpu(*s*)	*twl*	*tof*	cpu(*s*)	*twl*(%)	*tof*(%)	spd(x)	*twl*(%)	*tof*(%)	spd(x)
ibm01	76517	398	35.1	66006	189	25.0	63857	125	0.94	16.55	68.59	37.3	3.26	33.86243	26.6
ibm02	204734	492	58.9	178892	64	81.8	178261	3	2.34	12.93	99.39	25.2	0.35	95.3125	35.0
ibm03	185116	209	63.7	152392	10	61.8	150663	0	1.44	18.61	100.00	44.2	1.13	100	42.9
ibm04	196920	882	145.5	173241	465	94.3	172608	165	3.58	12.35	81.29	40.6	0.37	64.51613	26.3
*ibm05	420583	0	108.1	412197	0	191.4	413496	0	1.49	-	-	-	-	-	-
ibm06	346137	834	171.6	289276	35	131.8	286025	14	4.46	17.37	98.32	38.5	1.12	60	29.6
ibm07	449213	697	408.4	378994	309	218.8	379133	99	5.44	15.60	85.80	75.1	-0.04	67.96117	40.2
ibm08	469666	665	417.5	415285	74	199.0	412308	56	6.18	12.21	91.58	67.6	0.72	24.32432	32.2
ibm09	481176	505	673.1	427556	52	234.4	419199	47	4.74	12.88	90.69	142.0	1.95	9.615385	49.5
ibm10	679606	588	789.6	599937	51	467.9	598460	46	7.66	11.94	92.18	103.1	0.25	9.803922	61.1
Average										14.49	89.76	63.73	1.01	51.71	38.16

* The result comparison did not include the case ibm05 since it is a trivial case.