FLARE AWARE GLOBAL ROUTING OF STANDARD CELLS

By

PRITAM GAYEN

Under the guidance of

Dr. PRITHA BANERJEE

(Assistant Professor, University of Calcutta)

ABSTRACT

Extreme Ultraviolet Lithography (EUVL) is one of the most promising Next Generation Lithography (NGL) technologies [2]. Surface roughness of the optical system used in EUVL, leads to scattering of light (flare) from the surface. Distortion of the critical dimension (CD) is possible due to the high flare in the EUVL system which in turn worsens the CD uniformity of the layout. Now a day flare has become one of the most critical issues in EUVL.

In addition, the layout density non-uniformity and the flare periphery effect (the flare distribution at the periphery is much different from that in the center of a chip) also induce a large flare variation within a layout. Both of the high flare level and the large flare variation could worsen the control of critical dimension (CD) uniformity. Dummification (i.e., tiling or dummy fill) is one of the flare compensation strategies to reduce the flare level and the flare variation for the process with a clear-field mask in EUVL. But, it is costlier. It has been seen that the flare effect is higher in the center than the edge side of the chip-layout. Here we are trying to make a global design of a layout in a way so that the flare should be minimized.

INTRODUCTION

Extreme Ultraviolet Lithography (EUVL) is one of the most promising technologies since the ten times reduction in wavelength in EUVL offers the capability of a continuation of Moore's law beyond the 22 nm technology node. However, the used light of 13.5 nm wavelength is not transmitted, but absorbed by most of materials. Clear-field mask with photo-reflective martial is used to cope with this problem. Photo-reflective martial has uneven surface which causes undesired scattering of light from the surface. Moreover, layout patterns are formed by absorbers on the mask. Hence vacant regions are not covered by absorber and contribute to flare. Figure 1(a) shows the flare from a clear-field mask and Figure 1(b) shows the patterns on the clear-field mask. However, flare reduces the contrast between bright regions (vacant regions) and dark regions (layout patterns), and may result in critical dimension (CD) distortion. Since the flare is proportional to the surface roughness of the optical system and inversely proportional to the squared of wavelength [2], EUVL suffers from rather high level of flare compared to traditional lithography technologies. It is reported by the alpha demo tool (ADT) at IMEC that the intrinsic flare is about 16% [2]. On the other hand, the regions at the periphery

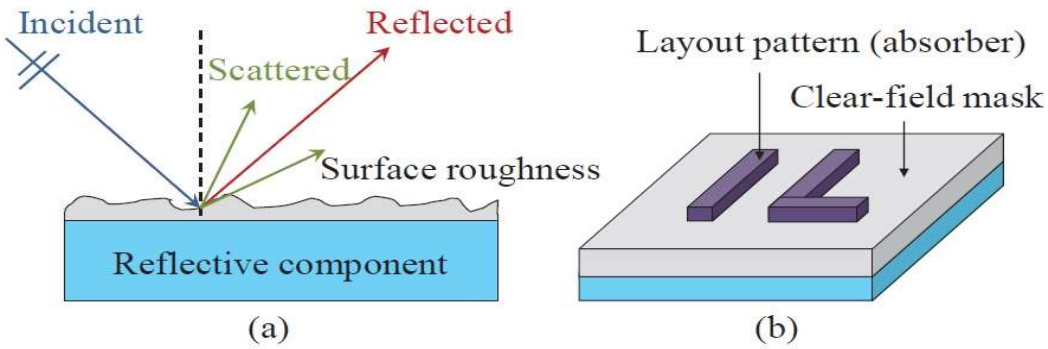


Figure 1: (a) Flare is undesired scattered light due to the surface roughness of reflective optical components and masks used in EUVL. (b) A clear-field mask on which layout patterns are formed by light absorbing materials.

of a chip receive much less flare compared to the regions in the center of a chip (assuming the regions outside the chip boundaries are dark-fields), causing a large flare variation. We refer to the phenomenon as *the flare periphery effect*. In addition, the non-uniformity of layout patterns may contribute to the flare variation within a chip as well. For the process with a clear-field mask, regions with lower pattern density contribute to more flare distribution than those with higher pattern density [2]. Since the high flare level causes CD distortion and the flare variation damages CD uniformity, flare compensation strategies are required.

There are two strategies for flare compensation. One is applying global CD resizing similar to optical proximity correction (OPC) on pattern features according to the flare value received by each feature [2]. However, previous work has reported that one percent change in flare level may cause 10 nm change in CD (CD sensitivity on flare is 10 nm/% Δflare level) at the 22 nm technology node and may be considerably larger for more advanced technology nodes [2]. As a result, a large flare variation may not be fully compensated by applying a global CD resizing and may cause the difficulty of controlling CD uniformity accurately. Another flare compensation strategy is dummification (i.e., tiling or dummy fill) [2]. By adding dummy patterns according to global flare distribution, intra-chip flare variation could be reduced. Although dummification may be limited to design rules and layout constraints, it has been shown that dummification can simultaneously reduce intra-chip flare level and flare variation in EUVL for the process with a clear field mask, and thus may greatly simplify the flare compensation methodology with global CD resizing [2].

Here we propose another approach for flare compensation. Flare density is modeled by the Gaussian density function. So we are trying to maintain the Gaussian density in the whole layout. That means we plot a Gaussian density function (3D) on the layout and trying to fill the chip layout by wire line (global design) maintaining the density given by that function everywhere on chip layout.

MOTIVATION

Traditionally used methods for flare reduction involves Dummification where amount of vacant region is decreased by putting an effective number of non-functional blocks (Dummy) which in turn reduces the flare. Dummy filling implies adding extra features into the mask as a result mask preparation cost can be increased in a large amount. Our objective is to reduce flare significantly without using Dummy fill. In our approach we divide the total layout area into some grids. We calculate density value for each grid from the Gaussian density function. We create a global route so that each grid of the layout area will satisfy the density constraint. If one grid has to fill up with density value D_g (say 0.89) and the total area of that grid is C_g (say 8 unit) then we multiply D_g and C_g. T_g is the target capacity up to which we have to fill that grid so that flare should be minimized. $T_g = D_g \times C_g$. ($T_g = 0.89 \times 8 = 7.12$).

PROBLEM FORMULATION

We are proposing a global routing method. To doing so we use Integer Linear Programming (ILP) procedure. Input for that global routing will be a set of net (N) and the total layout area (X x Y). X and Y is the number of tracks along X axis and Y axis respectively. Output of our method will be a global routing which will satisfy the density constraints.

We divide the total layout area into some grids. Let m is the no. of grids along X axis and n is the no. of grids along Y axis. [please note: The summetion of all grid area is greater than (X x Y) unit area.] Grids are squre in shape and let the total capacity of a grid g(m, n) is $C_{g(m,n)}$. $C_{g(m,n)}$ of each grid is not same, we will discuss this latter.

We calculate target density $T_{g(m,\,n)} = C_{g(m,n)} \times D_{g(m,\,n)}$ where $C_{g(m,n)}$ is the total capacity and $D_{g(m,\,n)}$ is the density calculated by the gaussian density function for each grid g(m, n).

We divide each net into 2 terminal subnets. Now we take all possible path between 2 terminals of every subnet. We assign a binary variable X_{ij} for each possible path(where i denotes the possible path number and j denotes the net number for X_{ij}).

Now we calculate the density $O_{ij}^{g(m,\,n)}$. If X_{ij} is selected by ILP then the capacity occupied by X_{ij} in grid g(m, n) is capacity $O_{ij}^{g(m,\,n)}$. So the ILP formulation will be –

Maximize Z.

$$Z = \sum_{j=1}^{n} \sum_{i=1}^{k} X_{ij}$$

Where i denotes one possible path between 2 terminals (k is the no. of possible paths for subnet n), j denotes the subnet no.(n is the total no. of subnets).

Subject to

$\sum_{i=1}^{k} X_{ij} \leq 1$ for all subnets 1 to n.

$$\sum_{j=1}^{n} \sum_{i=1}^{\mathrm{k}} \mathrm{O}_{\mathrm{ij}}^{\mathrm{g(m,n)}} \mathrm{X}_{\mathrm{ij}} \leq T_{g(m,n)}$$ for all grid g(m,n).

BACKGROUND

In this section we discuss the method of flare computation, pattern density and vacancy density.

COMPUTATION OF FLARE

In the previous works [2] it is reported that flare in EUVL can be modeled as a scattering point spread function (PSF). Flare distribution can be obtained by convolving the PSF with the original layout image intensity but directly convolving image intensity with PSF can be computationally expensive. To

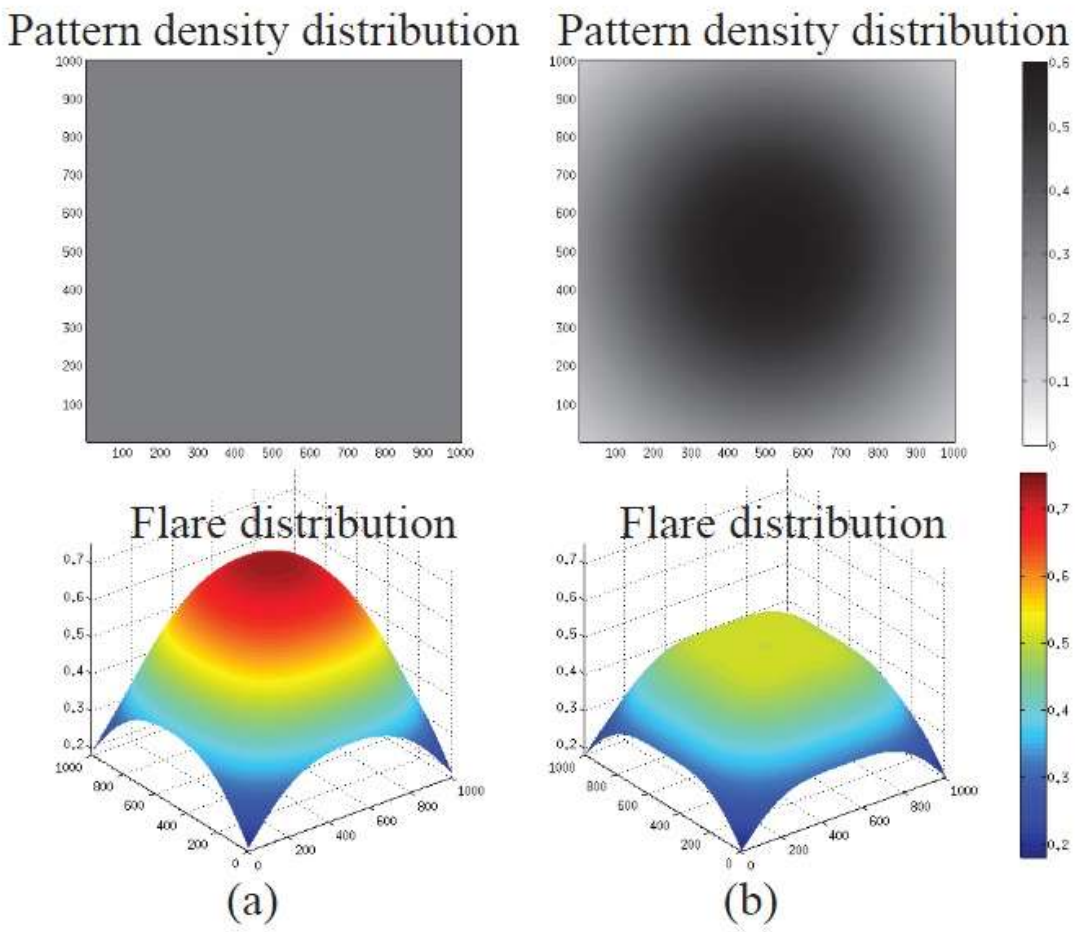


Figure 2: Flare comparison between layouts with different density distributions. (a) A layout with uniform density distribution may have large flare variation due to the flare periphery effect. (b) A layout with density distribution conforming to the global flare distribution has smaller flare variation.
snap shot from [2].

solve this problem entire layout is divided into same sized cells or grids and pattern density map for entire gridded layout can be an acceptable approximation of the image intensity. Vacant regions contributes more to the flare hence we have used vacancy density map for computation of flare. We have computed flare map F for gridded layout by using the following equation as in [2].

$$F(m, n) = D_v(m, n) \otimes PSF(m, n)$$

Where F(m,n), D_v(m,n) and PSF(m,n) are the flare, vacancy density and PSF for cell or grid (m,n).

PATTERN DENSITY AND VACANCY DENSITY

Pattern density for a cell or grid is defined as the ratio of actual area of patterns currently present in the cell or grid and total pattern area that can be allotted without

violating the design rule. Vacancy density map is computed as the (1-pattern density) for entire layout.

OVERALL METHODOLOGY

We randomly generate some nets. Then all nets are divided into 2 terminal subnets. Now we generate all possible paths for each 2 terminal subnets and assign a name X_{ij} for each path. These paths are generally I, L and Z type in shape. We divide the total layout area (X x Y) into some grids or cells g(m, n), where m is the no. of cells along X axis and n is the no. of cells along Y axis. These cells are square in shape. Now we calculate the density value $D_{g(m, n)}$ by the Gaussian density function [1] and calculate the total capacity $C_{g(m, n)}$ for each grid. We calculate the capacity $O_{ij}^{g(m, n)}$ occupied by each X_{ij} in each grid g(m, n). Now we formulate an ILP to solve from which we get which X_{ij} are 1. As X_{ij} are Boolean variable in our ILP formulation. Now we can design the global route for the given nets (taking randomly). At last we do the flare map F(m, n) for each grid of the generated global routing.

METHODOLOGY IN DETAIL

SORTING OF NET POINTS

We sort the net points according to Hamiltonian distance so that while we are routing a couple of points (subnets), we can choose a path of minimum wire length. We take the first point then put the point beside it which is closer to it according to Hamiltonian distance, chosen from the rest of the points in that particular net. Now we take the next point (recently chosen) and choose another point closer (Hamiltonian distance) to it.

ALGORITHM: SORTING OF NET POINTS

1. Do i=1 to N(N is the no. Of nets)
2. Do j=1 to last but one point of the corresponding net
3. Distance = no. of trackx + no. of tracky + 2 (taking as a maximum distance)
4. Target=0
5. Do k=1 to last point of this net
6. xd=|Xij - Xik|
7. yd=|Yij - Yik|
8. d=xd+yd
9. if Distance > k then
10. Distance=d
11. Targetk=k
12. k=k+1
13. If Targetk > 0 then(we have found a point for shifting)
14. Temp(x,y) = (x,y)ij+1
15. (x,y)ij+1=(x,y)i Targetk
16. (x,y)i Targetk = Temp(x,y)
17. j++

18. i++
19. end

GENERATION OF ALL POSSIBLE PATHS

We have 2 terminals say (x1, y1) and (x2, y2) as a subnet. We take a horizontal line x1 to x2 (if x1 ≠ x2) through the point (x1, y1). So the end point of this horizontal straight line will be (x2, y1). Now connect (x2, y1) with (x2, y2) by another straight line (vertical). The path is formed like 'L'. If we take the horizontal line through the point (x1, y1+1) then another end point of the straight line will be (x2, y1+1). Then we have to connect (x2, y1+1) & (x2, y2) by a vertical line and (x1, y1+1) & (x1, y1) by another vertical line. Now the path becomes like 'Z'. If we put (x1, y1+2) at the place of (x1, y1+1) and (x2, y1+2) at the place of (x2, y1+1) then we get another 'Z' path (this way we can get |y2-y1| no. of 'Z' paths). We can do the same for taking a vertical straight line and we can get |x2-x1| no. of 'Z' paths. When x1=x2 or y1=y2 then only a vertical or horizontal line can connect the points (x1, y1) and (x2, y2) then the shape looks like 'I'.

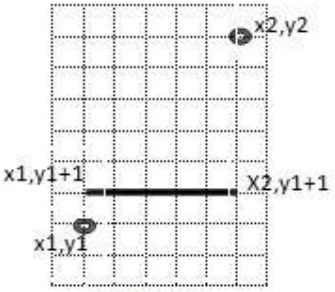


Diagram-1

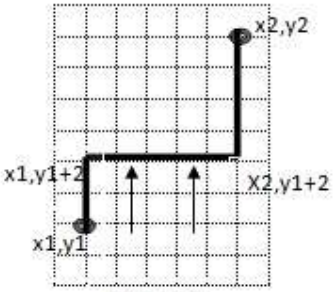


Diagram-3

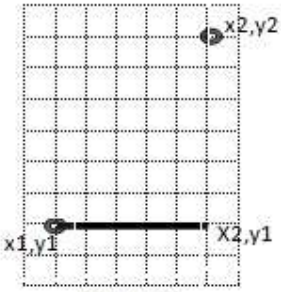


Diagram-5

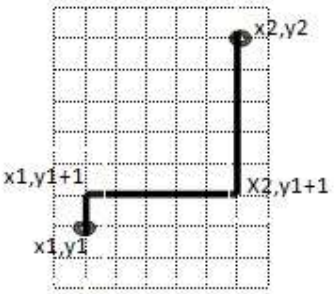


Diagram-2

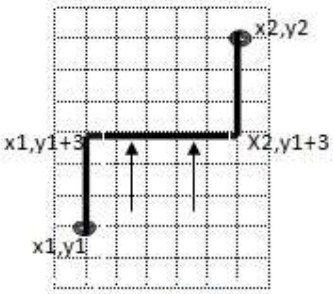


Diagram-4

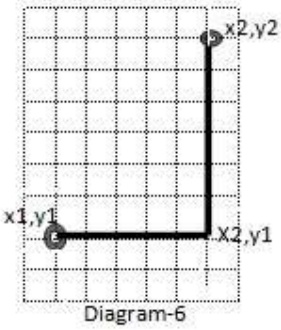


Diagram-6

Diagram-1,2 shows the creation of 'Z' shaped path.
Diagram-2,3,4 are 3 different 'Z' shaped path.
Diagram- 5,6 shows the creation of 'L' shaped path.
Diagram-7 shows the creation of 'I' shaped path.

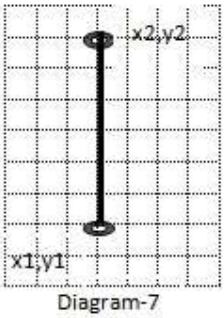


Diagram-7

ACCEPTANCE TEST FOR ALL X_{ij}

After generate a possible path X_{ij} we have to check whether it passes through any terminal of other subnets. If it passes through any terminal of other subnets then we discard the path. For this we superimpose a matrix (X x Y) over the total chip layout area. Initially we take a matrix (X x Y) and assign 1 value in which cell terminals are located and other cells are assigned with 0. After generating a path we check whether the path contains any 1 valued cell on the matrix except first and last point of that path. If we see any 1 valued cell in the matrix throughout the path then we assign 0 to its priority so that it could not be selected by ILP.

ALGORITHM: GENERATION OF ALL POSSIBLE PATHS

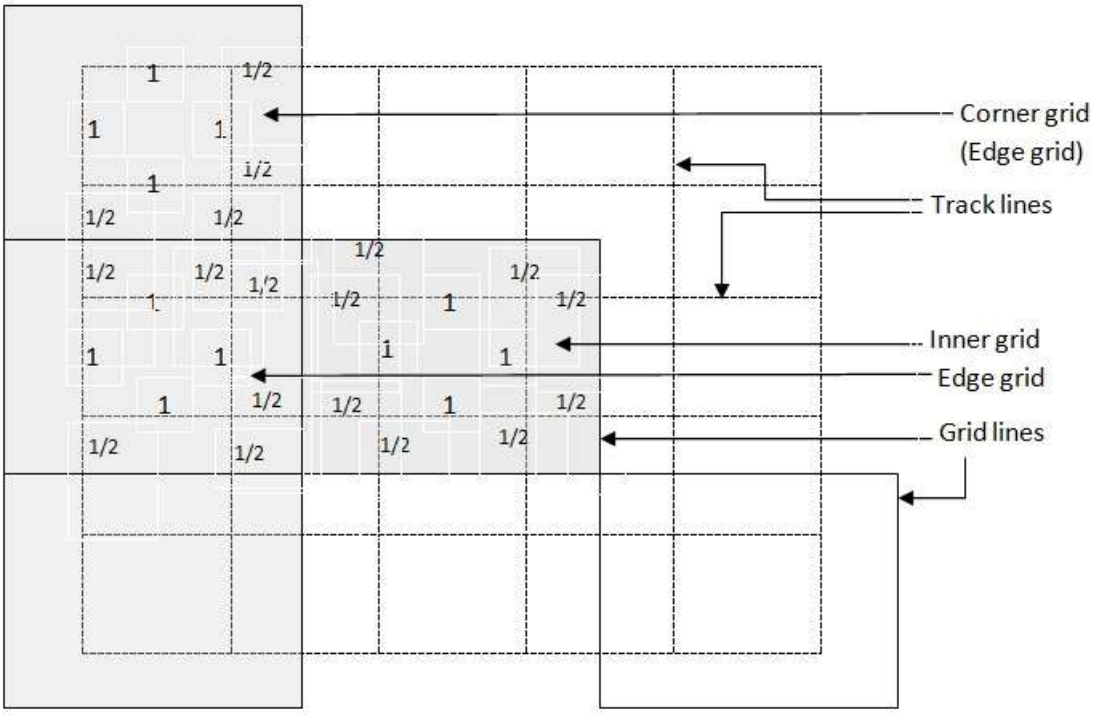
Given a subnet having two points (X1,Y1) and (X2,Y2), (X1, Y1),(X2,Y2), (X3,Y3),(X4,Y4) are 4 points for a possible path.

1. If x1=x2 or y1=y2 then
2. Priority=1(It is 'I' shaped path)
3. If shape is not valid then priority=0
4. X1=x1
5. Y1=y1
6. X2=x2
7. Y2=y2
8. Else if x1≠x2 and y1≠y2
9. i=0
10. Do until x1≠x2
11. X1=x1
12. Y1=y1
13. X2=x1+i
14. Y2=y1
15. X3=x1+i
16. Y3=y2
17. X4=x2
18. Y4=y2
19. If i=0 then Priority=0.99(it is 'L' shape)
20. Else priority=0.98(it is 'Z' shape)
21. If shape is not valid then priority=0
22. If x1>x2 then i=i-1
23. If x1<x2 then i=i+1
24. End loop
25. i=0
26. Do until y1≠y2
27. X1=x1
28. Y1=y1
29. X2=x1
30. Y2=y1+i
31. X3=x2
32. Y3=y1+i
33. X4=x2
34. Y4=y2

35. If i=0 then Priority=0.99(it is 'L' shape)
36. Else priority=0.98(it is 'Z' shape)
37. If shape is not valid then priority=0
38. If y1>y2 then i=i-1
39. If y1<y2 then i=i+1
40. End loop
41. end

CALCULATION OF THE TOTAL CAPACITY $C_{g(m, n)}$ FOR EACH GRID

We divide the total layout area (X x Y) into m x n grids. Each grid is square in shape and equal in size but the wire length capacity of each grid is not same. Inner grids (those are surrounded by other grids) have the maximum wire length capacity. Outer grids or edge grids have not maximum wire length capacity. Edge grids which are at corner of the layout have the lesser wire length capacity.



On the above diagram we can see 2 horizontal and 2 vertical wire lines in each grid. Inner grid have the wire length capacity (4 x 1) + (8 x 0.5) = 8 units. Outer grids or edge grids have the wire length capacity (4 x1) + (6 x 0.5) = 7 units where as the corner grids have (4 x 1) + (4 x 0.5) = 6 units of wire length capacity. If we consider that 't_r' is the no. of horizontal as well as vertical track inside a grid then wire length capacity of a inner grid will be $C_{g(m,n)} = 2t_r^2$. Wire length capacity of a outer grid will be $C_{g(m,n)} = 2t_r^2-(t_r \times 0.5)$ and corner grid will be $C_{g(m,n)} = 2t_r^2 - 2(t_r \times 0.5)$.

ALGORITHM: CALCULATION OF THE TOTAL CAPACITY $C_{g(m, n)}$ FOR EACH GRID

m is no. of tracks along X axis and n is no. of tracks along Y axis. t_r is the no. of tracks along X as well as Y axis.

1. Do for all grid g(m,n)
2. If (i=1 & j=1) or (i=m & j=n) or (i=1 & j=n) or (j=1 & i=m) then
3. Capacity=$2t_r^2 - 2t_r$ x 0.5
4. Else if (i=1 or j=1)
5. Capacity=$2t_r^2 - t_r$ x 0.5
6. Else
7. Capacity=$2t_r^2$
8. end

CALCULATION OF THE DENSITY VALUE $D_{g(m, n)}$

We use a Gaussian density function for calculation of target density value $D_{g(m ,n)}$ for each grid g(m, n). The function is:

$$f(x, y) = A \exp\left(-\left(\frac{(x - x_0)^2}{2\sigma_x^2} + \frac{(y - y_0)^2}{2\sigma_y^2}\right)\right)$$

Here A is the amplitude x_0 , y_0 is the center σ_x , σ_y are the spreads along x and y respectively. We put A = 1, x_0 = 0, y_0 = 0, σ_x is no. of grids along X axis, σ_y is no. of grids along Y axis.

CALCULATION OF THE CAPACITY $O_{ij}^{g(m, n)}$

$O_{ij}^{g(m, n)}$ is the capacity of X_{ij} in grid g(m, n). That means how much wire length will occupied of grid g(m, n) if path X_{ij} is selected for routing. For finding this we calculate all the points of the possible path X_{ij} and put them in an array of size X+Y (the array size is X+Y as this will be the maximum path length). Now we take points one by one from that array and check whether it is inside the grid g(m, n). if we found a point inside the grid then we count how many points are inside that grid of that possible path. This process will end when we found a point outside of that grid. If first point of the path is inside the grid then -

$O_{ij}^{g(m, n)}$ = 0.5 + no. of points in the grid g(m, n) - 1 + 0.5

If first point of the path is not inside the grid then -

$O_{ij}^{g(m, n)}$ = no. of points in the grid g(m, n) - 1 + 0.5

ALGORITHM: CALCULATION OF THE CAPACITY $O_{ij}^{g(m, n)}$

t_r is the no. tracks per grid.

For each grid g(m,n) do for all X_{ij}

1. count=0
2. j=0
3. do until we reach the end point of the array
4. if m = (X_j/t_r + X_j modulo t_r) and n = (Y_j/t_r + Y_j modulo t_r) then
5. count=count+1
6. end loop
7. wirelength=count-1;
8. if m≠($X_{last\ point}/t_r$ + $X_{last\ point}$ modulo t_r) and n ≠ ($Y_{last\ point}/t_r$ + $Y_{last\ point}$ modulo t_r) then
9. wirelength=wirelength+0.5
10. if m≠($X_{1st\ point}/t_r$ + $X1_{st\ point}$ modulo t_r) and n ≠ ($Y_{1st\ point}/t_r$ + $Y_{1st\ point}$ modulo t_r) then

11. wirelength=wirelength+0.5
12. j=j+1
13. End

TOTAL CODING AND SOFTWARES

We use C language to write the source code. To solve the ILP formulation we use lp_solve. And for calculating flare map we use scilab.

The source code is given bellow:

```
#include<stdio.h>
#include <stdlib.h>
#include <time.h>
#include<math.h>


struct shape//L Z or I shape
{
	long x1,y1, x2,y2, x3,y3, x4,y4;
	float priority;

};

struct shape *MAT[1000][1000];//holds the address of shapes
long NETMAT[1000][1000];
struct grid
{
	float Tcapacity;
	struct grid_seg *GS;
};

struct grid_seg//holds how much part of a shape is in a grid
{
	float capacity;
	long mati,matj;
	struct grid_seg *nxt;
};

long randum(long i, long j)
{

	long r=rand();
	r=r%j;
	return(r+i);
```

```
}


struct grid_seg * density_find(long X, long Y, long k, long m, long n, long tr)
{

	long p,q,i,j;
	long arrx[X+Y],arry[X+Y],count,xx1,yy1,xx2,yy2;
	float wirelength;
	struct grid_seg *head=NULL,*ptr,*data;
	for(p=1;p<k;p++)
	{
	for(q=1;MAT[p][q]!=NULL;q++)
	{
		//points insertion on matrix
		i=0;
		xx1=MAT[p][q]->x1;
		yy1=MAT[p][q]->y1;
		xx2=MAT[p][q]->x2;
		yy2=MAT[p][q]->y2;
		if(xx1!=xx2 && yy1==yy2)
		{
			for(;xx1!=xx2;i++)
			{
				if(xx1>xx2)
					xx1--;
				if(xx1<xx2)
					xx1++;
				arrx[i]=xx1;
				arry[i]=yy1;
			}
			arrx[i]=xx1;
			arry[i]=yy1;
			i++;
		}
		else
		if(xx1==xx2 && yy1!=yy2)
		{

			for(;yy1!=yy2;i++)
			{
				if(yy1>yy2)
					yy1--;
				if(yy1<yy2)
					yy1++;
				arrx[i]=xx1;
				arry[i]=yy1;
```

```
		}
		arrx[i]=xx1;
		arry[i]=yy1;
		i++;
}

xx1=MAT[p][q]->x2;
yy1=MAT[p][q]->y2;
xx2=MAT[p][q]->x3;
yy2=MAT[p][q]->y3;
if(xx1!=xx2 && yy1==yy2)
{
		for(;xx1!=xx2;i++)
		{
			if(xx1>xx2)
				xx1--;
			if(xx1<xx2)
				xx1++;
			arrx[i]=xx1;
			arry[i]=yy1;
		}
		arrx[i]=xx1;
		arry[i]=yy1;
		i++;
}
else
if(xx1==xx2 && yy1!=yy2)
{

		for(;yy1!=yy2;i++)
		{
			if(yy1>yy2)
				yy1--;
			if(yy1<yy2)
				yy1++;
			arrx[i]=xx1;
			arry[i]=yy1;
		}
		arrx[i]=xx1;
		arry[i]=yy1;
		i++;
}

xx1=MAT[p][q]->x3;
yy1=MAT[p][q]->y3;
xx2=MAT[p][q]->x4;
yy2=MAT[p][q]->y4;
```

```
if(xx1!=xx2 && yy1==yy2)
{
	for(;xx1!=xx2;i++)
	{
		if(xx1>xx2)
			xx1--;
		if(xx1<xx2)
			xx1++;
		arrx[i]=xx1;
		arry[i]=yy1;
	}
	arrx[i]=xx1;
	arry[i]=yy1;
	i++;
}
else
if(xx1==xx2 && yy1!=yy2)
{

	for(;yy1!=yy2;i++)
	{
		if(yy1>yy2)
			yy1--;
		if(yy1<yy2)
			yy1++;
		arrx[i]=xx1;
		arry[i]=yy1;
	}
	arrx[i]=xx1;
	arry[i]=yy1;
	i++;
}
//points insertion on matrix end

//capacity of a grid
count=0;
for(j=0;j<i;j++)
{
	if(m==(arrx[j]/tr)+(arrx[j]%tr) && n==(arrx[j]/tr)+(arrx[j]%tr))
	{
		count++;
	}
	wirelength=count-1;
	if(m==(arrx[0]/tr)+(arrx[0]%tr) && n==(arrx[0]/tr)+(arrx[0]%tr))
	{
		wirelength+=0.5;
	}
```

```
                        if(m==(arrx[i-1]/tr)+(arrx[i-1]%tr) && n==(arrx[i-1]/tr)+(arrx[i-1]%tr))
                        {
                                wirelength+=0.5;
                        }
                        if(count==0)
                                wirelength=0;

                }
                //capacity of a grid end

                data=(struct grid_seg *)malloc(sizeof(struct grid_seg));
                data->capacity=wirelength;
                data->mati=p;
                data->matj=q;
                data->nxt=NULL;

                if(head==NULL)
                {
                        head=data;
                        ptr=head;
                }
                else
                {
                        ptr->nxt=data;
                        ptr=ptr->nxt;
                }


        }
        }



        return(head);
}




int check_shape(long X,long Y,long x1,long y1,long x2,long y2, long x3,long y3,long x4,long
y4)
{
        long arrx[X+Y],arry[X+Y],i,j,xx1,xx2,yy1,yy2,temp=1,diff;
        i=0;
        xx1=x1;
        yy1=y1;
```

```
xx2=x2;
yy2=y2;
if(xx1!=xx2 && yy1==yy2)
{
for(;xx1!=xx2;i++)
{
        if(xx1>xx2)
                xx1--;
        if(xx1<xx2)
                xx1++;
        arrx[i]=xx1;
        arry[i]=yy1;
}
arrx[i]=xx1;
arry[i]=yy1;
i++;
}
else
if(xx1==xx2 && yy1!=yy2)
{

for(;yy1!=yy2;i++)
{
        if(yy1>yy2)
                yy1--;
        if(yy1<yy2)
                yy1++;
        arrx[i]=xx1;
        arry[i]=yy1;
}
arrx[i]=xx1;
arry[i]=yy1;
i++;
}

xx1=x2;
yy1=y2;
xx2=x3;
yy2=y3;
if(xx1!=xx2 && yy1==yy2)
{
for(;xx1!=xx2;i++)
{
        if(xx1>xx2)
                xx1--;
        if(xx1<xx2)
                xx1++;
```

```
        arrx[i]=xx1;
        arry[i]=yy1;
}
arrx[i]=xx1;
arry[i]=yy1;
i++;
}
else
if(xx1==xx2 && yy1!=yy2)
{

for(;yy1!=yy2;i++)
{
        if(yy1>yy2)
                yy1--;
        if(yy1<yy2)
                yy1++;
        arrx[i]=xx1;
        arry[i]=yy1;
}
arrx[i]=xx1;
arry[i]=yy1;
i++;
}

xx1=x3;
yy1=y3;
xx2=x4;
yy2=y4;
if(xx1!=xx2 && yy1==yy2)
{
for(;xx1!=xx2;i++)
{
        if(xx1>xx2)
                xx1--;
        if(xx1<xx2)
                xx1++;
        arrx[i]=xx1;
        arry[i]=yy1;
}
arrx[i]=xx1;
arry[i]=yy1;
i++;
}
else
if(xx1==xx2 && yy1!=yy2)
{
```

```
	for(;yy1!=yy2;i++)
	{
		if(yy1>yy2)
			yy1--;
		if(yy1<yy2)
			yy1++;
		arrx[i]=xx1;
		arry[i]=yy1;
	}
	arrx[i]=xx1;
	arry[i]=yy1;
	i++;
	}

	for(j=1;j<(i-1);j++)
	{
	if(NETMAT[arrx[j]][arry[j]]==1)
		temp=0;
	}
	return(temp);

}


void shape_find(long x1, long y1, long x2, long y2, long mati, long X, long Y)
{
	long matj=1,i,diff,temp;
	struct shape *ILZ;

	if(x1==x2 || y1==y2)
	{
	ILZ=(struct shape *)malloc(sizeof(struct shape));
	ILZ->x1=x1;
	ILZ->y1=y1;
	ILZ->x2=x2;
	ILZ->y2=y2;
	ILZ->x3=x2;
	ILZ->y3=y2;
	ILZ->x4=x2;
	ILZ->y4=y2;
	ILZ->priority=1.00;//priority for I shape
	temp=check_shape(X,Y,ILZ->x1,ILZ->y1,ILZ->x2,ILZ->y2,ILZ->x3,ILZ->y3,ILZ->x4,ILZ-
>y4);
	if(temp==1)
	{
		MAT[mati][matj]=ILZ;
```

```
                matj++;
        }


        }
        else
        if(x1!=x2 && y1!=y2)
        {
        for(i=0;(x1+i)!=x2;)//shifting vertical line
        {
                ILZ=(struct shape *)malloc(sizeof(struct shape));

                ILZ->x1=x1;
                ILZ->y1=y1;
                ILZ->x2=x1+i;
                ILZ->y2=y1;
                ILZ->x3=x1+i;
                ILZ->y3=y2;
                ILZ->x4=x2;
                ILZ->y4=y2;

                if(i==0)
                        ILZ->priority=0.99;//priority for L shape
                else
                        ILZ->priority=0.98;//priority for Z shape
                temp=check_shape(X,Y,ILZ->x1,ILZ->y1,ILZ->x2,ILZ->y2,ILZ->x3,ILZ->y3,ILZ-
>x4,ILZ->y4);
                if(temp==1)
                {
                        ILZ->priority=0.00;
                }
                MAT[mati][matj]=ILZ;
                matj++;

                if(x1>x2)
                        i--;
                if(x1<x2)
                        i++;
        }

        for(i=0;(y1+i)!=y2;)//shifting horizontal line
        {
                ILZ=(struct shape *)malloc(sizeof(struct shape));

                ILZ->x1=x1;
                ILZ->y1=y1;
                ILZ->x2=x1;
```

```
                ILZ->y2=y1+i;
                ILZ->x3=x2;
                ILZ->y3=y1+i;
                ILZ->x4=x2;
                ILZ->y4=y2;

                temp=1;
                temp=check_shape(X,Y,ILZ->x1,ILZ->y1,ILZ->x2,ILZ->y2,ILZ->x3,ILZ->y3,ILZ-
>x4,ILZ->y4);
                if(temp==0)
                {
                        ILZ->priority=0.00;
                }
                else
                {
                        if(i==0)
                                ILZ->priority=0.99;//priority for L shape
                        else
                                ILZ->priority=0.98;//priority for Z shape


                }
                MAT[mati][matj]=ILZ;
                matj++;

                if(y1>y2)
                        i--;
                if(y1<y2)
                        i++;

        }


        }

        MAT[mati][matj]=NULL;
}

void net(long X,long Y,long NET,long tr)
{

        long ARR1[X][Y],len[NET];
        long no_of_points,i,j,valid_x,valid_y;
        long netx[NET][9],nety[NET][9];

        long distance,targetk,k,xd,yd,d,tempx,tempy;
        FILE *out;
```

```
long no_of_gridy,no_of_gridx;

//initialization
for(i=0;i<=X;i++)
for(j=0;j<=Y;j++)
{	ARR1[i][j]=(-1);
	NETMAT[i][j]=0;
}

//initialization end

out=fopen("TEST_FLARE.txt","w");
fprintf(out," X = %ld, Y = %ld, track = %ld",X,Y,tr);
//point insertion
fprintf(out,"\n INSERTED POINTS\n-----------------------------------\n");
srand ( time(NULL) );
for(i=0;i<NET;i++)
{
while(1)
{
	no_of_points=rand();
	no_of_points=no_of_points%8;// numeric is the upper bound of points
	if(no_of_points>1)// numeric is the lower bound of points
		break;
}
len[i]=no_of_points;
for(j=0;j<no_of_points;j++)
{
	while(1)
	{
		valid_x=randum(1,X*Y);
		//valid_y=randum(1,Y);
		valid_y=(valid_x/X)+1;
		valid_x=(valid_x%Y)+1;

		if(ARR1[valid_x][valid_y]<0)//arr[x][y]<0 means it is -1 means not occupied
		{
			ARR1[valid_x][valid_y]=i+1;//net no. is inserted in the place arr[x][y]
			NETMAT[valid_x][valid_y]=1;
			break;
		}
	}
	netx[i][j]=valid_x;
	nety[i][j]=valid_y;
```

```
        fprintf(out," [ %ld,%ld ]",netx[i][j],nety[i][j]);
}

fprintf(out,"\n");
}
//point insertion end


//sorting
fprintf(out,"\n\n");
fprintf(out,"\n POINTS AFTER SORTING\n-----------------------------------\n");
for(i=0;i<NET;i++)
{
for(j=0;(j+2)<len[i];j++)
{
        distance=(X+Y)+2;
        targetk=0;
        for(k=j+1;k<len[i];k++)
        {
                xd=netx[i][j]-netx[i][k];
                yd=nety[i][j]-nety[i][k];
                if(xd<0)
                        xd=xd*(-1);
                if(yd<0)
                        yd=yd*(-1);
                d=xd+yd;
                if(d<0)
                        d=d*(-1);
                if(distance > d)
                {
                        distance=d;
                        targetk=k;
                }

        }
        if(targetk>0)
        {
                tempx=netx[i][j+1];
                tempy=nety[i][j+1];

                netx[i][j+1]=netx[i][targetk];
                nety[i][j+1]=nety[i][targetk];

                netx[i][targetk]=tempx;
                nety[i][targetk]=tempy;
        }
```

```
}
}

//display netx nety

for(i=0;i<NET;i++)
{
for(j=0;j<len[i];j++)
{
        fprintf(out," [%ld,%ld]",netx[i][j],nety[i][j]);

}
fprintf(out,"\n");
}
fprintf(out,"\n\n");
//display end

//sorting end


//shape finding
k=1;
for(i=0;i<NET;i++)
{
for(j=0;j<(len[i]-1);j++,k++)
{
        shape_find(netx[i][j],nety[i][j],netx[i][j+1],nety[i][j+1],k,X,Y);

}
}
//shape finding end


//dispaly shape
long c=0;
for(i=1;i<k;i++)
{
for(j=1;MAT[i][j]!=NULL;j++,c++)
{
        fprintf(out,"\nX%ld_%ld        %.2f\n-------------------------------------\n %ld %ld\n
%ld %ld\n %ld %ld\n %ld %ld\n\n\n ",i,j,MAT[i][j]->priority,MAT[i][j]->x1,MAT[i][j]-
>y1,MAT[i][j]->x2,MAT[i][j]->y2,MAT[i][j]->x3,MAT[i][j]->y3,MAT[i][j]->x4,MAT[i][j]->y4);
}
}
fprintf(out,"\n total vaiables = %ld\n\n",c);
//display shape end
```

```
	//density finding

	no_of_gridx=X/tr;
	no_of_gridy=Y/tr;
	struct grid *GMAT[no_of_gridx][no_of_gridy];


	for(i=1;i<=no_of_gridx;i++)
	{
	for(j=1;j<=no_of_gridy;j++)
	{
		//total density of a grid
		if((i==1 && j==1) || (i==no_of_gridx && j==no_of_gridy) || (i==1 && j==no_of_gridy) || (i==no_of_gridx && j==1))
		{
			GMAT[i][j]=(struct grid *)malloc(sizeof(struct grid));
			GMAT[i][j]->Tcapacity=(2*tr*tr)-2*(tr*0.5);//total capacity of a inner grid & no. of track unit less in an corner edge.
		}
		else
		{
			if((i==1 && j!= 1 && j!=no_of_gridy) || (j==1 && i!= 1 && i!=no_of_gridx))
			{
				GMAT[i][j]=(struct grid *)malloc(sizeof(struct grid));
				GMAT[i][j]->Tcapacity=(2*tr*tr)-(tr*0.5);// if grid is not inner & not corner then tr track unit will less from capacity.
			}
			else
			{
				GMAT[i][j]=(struct grid *)malloc(sizeof(struct grid));
				GMAT[i][j]->Tcapacity=2*tr*tr;//if grid is inner grid then the total capacity.
			}
		}
		//total density of a grid end



		GMAT[i][j]->GS=density_find(X,Y,k,i,j,tr);//find the density of a shape in a grid

	}
	}

	//gaussian density

	float gaussMAT[no_of_gridx][no_of_gridy];
```

```
float a=1,density;
float sigmax=no_of_gridx,sigmay=no_of_gridy;


for(i=(no_of_gridx/2)*(-1);i<=(no_of_gridx/2);i++)
{
for(j=(no_of_gridy/2)*(-1);j<=(no_of_gridy/2);j++)
{
        density=(-0.5)*((2*i/sigmax)*(2*i/sigmax) + (2*j/sigmay)*(2*j/sigmay));
        density=a*expf( density );
        gaussMAT[i+1+(no_of_gridx/2)][j+1+(no_of_gridy/2)]=density;

}

}
//gaussian density end

//density finding end
fclose(out);

//problem formulation
struct grid_seg *ptr;
out=fopen("problem_formulation.lp","w");

fprintf(out,"\n\n/*expression*/\n\n");
fprintf(out," max: ");
for(i=1;i<=k;i++)
{
for(j=1;MAT[i][j]!=NULL;j++)
{
        if(MAT[i][j]->priority!=0)
                fprintf(out,"+ %f*X%ld_%ld ",MAT[i][j]->priority,i,j);
}
}
fprintf(out,";\n\n\n");


fprintf(out,"\n\n/*subject to */\n \n\n");
for(i=1;i<k;i++)
{
if(MAT[i][1]!=NULL)
{
        fprintf(out,"  con%ld:  ",i);
        for(j=1;MAT[i][j]!=NULL;j++)
        {
                fprintf(out,"+ X%ld_%ld ",i,j);
        }
```

```
                fprintf(out," <= 1;\n\n");
        }
        }
        for(i=1;i<=no_of_gridx;i++)
        {
        for(j=1;j<=no_of_gridy;j++)
        {
                fprintf(out,"  con%ld_%ld: ",i,j);
                ptr=GMAT[i][j]->GS;
                for(;ptr!=NULL;ptr=ptr->nxt)
                {
                                fprintf(out,"+    %f    X%ld_%ld    ",ptr->capacity,ptr->mati,ptr-
>matj);
                }
                //fprintf(out," <= %f ;\n\n",GMAT[i][j]->Tcapacity);
                fprintf(out," <= %f ;\n\n",GMAT[i][j]->Tcapacity*gaussMAT[i][j]);
        }
        }
        fprintf(out,"\n \n");



        fprintf(out,"\n\n/*variables*/\n\n");
        for(i=1;i<=k;i++)
        {
        for(j=1;MAT[i][j]!=NULL;j++)
        {
                fprintf(out,"bin X%ld_%ld;\n",i,j);
        }
        }
        fprintf(out," ");
        fclose(out);
        //problem formulation end

}


long char_to_num(char num[])
{
        long i,num1,num2;
        for(i=0,num1=0;num[i]!='\0';i++)
        {
        num2=num[i];
        num2-=48;
        num1=num1*10+num2;

        }
```

```
        return(num1);
}


void main(int arg,char *num[])
{

        long X,Y,NET,tr;
        X=char_to_num(num[1]);
        Y=char_to_num(num[2]);
        NET=char_to_num(num[3]);
        tr=char_to_num(num[4]);
        net(X,Y,NET,tr);
        //system("lp_solve -s problem_formulation.lp > output.txt");
        //collectXij();
}
```

EXPERIMENTAL RESULT

Sl. No.	No. of nets	Using Uniform density	Guided by Gaussian density function	Using Uniform density (Flare in standard deviation)	Guided by Gaussian density function (Flare in standard deviation)
		no. of connected nets	no. of connected nets		
1	2	3 of 3	3 of 3	0.22	0.22
2	2	3 of 3	3 of 3		
3	2				
4	2				
5	2				
6	2				
7	2				

CONCLUSION

As it is a global routing (which will minimize the flare) so after routing we need not insert dummy fill that is why it is efficient than dummification. Again it is useful than perturbation technique of flare minimization as we need not route more than one times.

Here we give 50 examples and comparing with uniform global routing and global routing guided by Gaussian density function. As we are using a PC the computational time required is more so we take upto 20 nets in 20 x 20 layout area. For a faster machine it is possible to take large value as nets and greater layout area.

FUTURE WORK

Here we create some nets randomly to route by our algorithm. Now we have to move for some standard circuit layout nets from the real world.

REFERENCES

[1] http://en.wikipedia.org/wiki/Gaussian_function

[2] Shao-Yun Fang and Yao-Wen Chang "Simultaneous Flare Level and Flare Variation Minimization with Dummification in EUVL", Proc. of the 49th Annual Design Automation Conference (DAC), pp. 1179-1184, 2012