VLSI Module Placement with Pre-placed Modules and Considering Congestion Using Solution Space Smoothing*

Sheqin Dong[1] Xianlong Hong[1] Xin Qi [1] Ruijie Wang [1] Song Chen[1] Jun Gu[2]

[1]Department of Computer Science and Technology, Tsinghua University, Beijing, P.R.C., 100084
Tel:+86-010-62785564 Fax:+86-010-62781489 Email:dongsq@mail.tsinghua.edu.cn
[2]Department of Computer Science, Science & Technology University of Hong Kong , Email: gu@cu.ust.hk

Abstract: Solution space smoothing allows a local search heuristic to escape from a poor, local minimum. In this paper, we propose a technique that can smooth the rugged terrain surface of the solution space of a placement problem. We design the solution space smoothing algorithms for VLSI placement with pre-placed modules and placement with consideration of congestion. Experiment results demonstrated that solution space smoothing is very efficient for VLSI module placement, and it can be applied to all floorplanning representations proposed so far.

I. Introduction

Due to the rugged terrain surface of the search space of a placement, it often gets stuck at a locally optimum configuration. In recent years, with the break through in floorplanning representations, simulated annealing, which can escape from a locally optimum configuration, were widely used for VLSI module placement.

To use a stochastic optimization algorithm to search an optimal placement for a placement instance, the first thing is to devise a proper coding scheme. For general placement or floorplan[1], several coding schemes were recently proposed, namely, Sequence-Pair (SP)[2], O-tree [3], Bound–Sliceline -Grid (BSG)[4], Corner Block List(CBL)[5][6] .

To make the coding scheme based placement algorithm more efficient and to reduce the effect of local minimum point, we introduce a solution space smoothing method for placement in this paper. All conducted research works on smoothing heuristics so far are on Travelling Salesman Problem. As we know, there has been no similar work for module placement so far. Solution space smoothing is a special technique of multi-space search developed in recent years [7][8]. In paper [7], Jun Gu and Xiaofei Huang had applied this method to the classical NP-hard problem: traveling salesman problem (TSP). Paper [8] investigates this method for traveling salesman problem thoroughly.

The basic idea of the proposed method for VLSI placement is simple. Initially, a simplified placement instance with a smooth terrain surface is solved. Then, a more complicated placement instance that has a rougher terrain surface is generated. It takes the solution of the previously solved placement as an initial placement and further improves the placement. Eventually, the original placement instance with the most complicated search space structure is solved. The solutions of the simplified problem instances are used to guide the search of more complicated ones. Compared with simulated annealing algorithm, solution space smoothing method needs few control parameters and the parameters are easy to determine. For placement including soft blocks, i.e. floorplanning, the time complexity of our method are nearly the same as placement including only hard blocks. Whereas a simulated annealing based algorithm will increase greatly because of soft blocks

We test the performance of this method using the general floorplan representation, Bounded Slice-line Grid (BSG) [4]. We use BSG only for it could be easily understood. Our method can be applied to all floorplanning representations proposed so far. We design algorithms for placement with pre-placed modules and placement with considering congestion. The MCNC benchmarks are used in the test. Experiment results prove that it is a new efficient optimization algorithm for VLSI module placement.

The rest of the paper is composed as follows: Sec. II describes the principle of placement with solution space smoothing method, the placement algorithm and its implementation is also presented. Sec. III gives the experimental results. Sec.IV is the conclusions.

II. VLSI Placement Using Solution Space Smoothing

A. Problem

Given a set of circuit building blocks, the placement problem can be defined as follows. A set $M=\{M_1, M_2, \ldots, M_n\}$ of rectangular blocks lie parallel to the coordinate axes in the first quadrant of the plane. A set of nets specifying the interconnections between pins of blocks and a set of pads (external pins) are also given. A placement $P=\{Mi\ (x_i, y_i) \mid 1 <= i <= n\}$ is an assignment of coordinates to the lower left corners of n rectangular blocks such that there is no two rectangular blocks overlapping. Each M_i is defined by a tuple (h_i , w_i) , where h_i and w_i are the height and the width of block M_i, respectively . The objective of the placement is to find an assignment so that the chip area and interconnection wire-length between blocks are minimized while satisfying the given constraints, if any.

When all circuit blocks have fixed area, fixed length and width, the assignment process is called placement. When some of the circuit blocks have fixed area while their width and length can vary within a specific range, the assignment

* This work is supported by the National Natural Science Foundation of China 60121120706 and National Natural Science Foundation of USA CCR-0096383, the National Foundation Research(973) Program of China G1998030403, the National Natural Science Foundation of China 60076016 and 863 Hi-Tech Research & Development Program of China 2002AA1Z1460

process is called floorplanning.

B. Solution Space Smoothing for A Placement Instance

Based on the size changing, we can transform the original placement instance into a series of placement instances. This transformation process is the solution space smoothing process of a placement instance. Assume Pi is the original placement instance with n blocks, Pi^0 is a placement instance with n blocks and all blocks of Pi^0 have the same size as the smallest block in Pi. Pi^1 is the instance with one block chooses from Pi and (n-1) blocks choose from Pi^0. So Pi^i has i blocks from Pi and (n-i) blocks from Pi^0. Therefore, Pi = Pi^n . Instead of restore blocks sizes one by one from the start point Pi^0, we adopt another smoothing strategy: from Pi^0, we slightly change the sizes of all the blocks in Pi^0 simultaneously and produce Pi^1, in the same way, from Pi^1 we produce Pi^2 , and so on. This also can produce a smoothed sequence Pi^0, Pi^1 , Pi^2 , Pi^3 ,Pi^4 …. in the sense of size.

So the assignment process of circuit building blocks using solution space smoothing method can be used not only for placement but also for floorplanning because of the size changing in the sequence Pi^0, Pi^1 , Pi^2 , Pi^3 ,Pi^4 ….

A simple placement instance, without considering the nets among blocks, is that all the circuit building blocks to be placed on a chip have the same size. In this case, the solution space is flattened since there are much less local minimum points in the solution space. We can use it as an initial smoothed solution space for a placement instance that has the same number of circuit blocks. In practice, we set the initial placement instance with all circuit blocks have the same size as formula (1) and (2). Every pin on the blocks will have its new position. Assume that, on block M_i, the *jth* original pin position is ($PinX^j$, $PinY^j$), after the smooth operation that all blocks were reduced to the same size, the new pin position can be calculated by formula (3).

$$\bar{w} = \min\{w_1,...,w_i,...,w_n\} \quad (1) \qquad (\bar{w},\bar{h}) \quad (2) \qquad \begin{matrix} PinX^j * (\bar{w}/w_i) \\ PinY^j * (\bar{h}/h_i) \end{matrix} \quad (3)$$
$$\bar{h} = \min\{h_1,...,h_i,...,h_n\}$$

A series of simplified placement instances can be created by a specified smoothing factor, α. To keep the similarity between instances in the series, we require the sizes of blocks changed in a slightly, gradually and monotonously increasing mode. After apply a smoothing function S(α), The size of *ith* block can be derived by applying formula (4), the new pin position can be calculated by formula (5).

$$S(\alpha): \begin{cases} w_i(\alpha) = \bar{w} + (w_i - \bar{w})^{\alpha} \\ h_i(\alpha) = \bar{h} + (h_i - \bar{h})^{\alpha} \end{cases} (4) \qquad \begin{matrix} PinX^j * (w_i(\alpha)/w_i) \\ PinY^j * (h_i(\alpha)/h_i) \end{matrix} \quad (5)$$

To guarantee the sizes of blocks changed in a slightly, gradually and monotonously increasing mode, we normalize items in formula S(α) to the range (0, 1).

When α is decreased from a large number to 1, a series of simplified placement instances are generated according to formula S(α). A solution space generated from a larger α exhibits a smoother terrain surface, and a solution space generated from a smaller α exhibits a more rugged terrain surface. When α>>1, each block size will be reduced to the initial one; when α = 1, each block will be of its original size.

C. Algorithm

We use BSG coding scheme and its neighborhood solution producing strategies to search an optimal or nearly optimal solution under certain α value using the cost function (*area*+λ**total_wirelength*). Where *area* is the area of the smallest rectangle enclosing the placement, *total_wirelength* is the sum of all the net length using the half perimeter model, while λ is the weight. Start from the initial placement instance that has a fairly "flat" solution space, and find the solution to this simplified problem using local search algorithm. A BSG based local search algorithm, BSG_local_search, could be easily derived.

We summarize the placement process based on solution space smoothing as the algorithm VP_SSS(). We combine clustering, incremental optimization, placement with empty rooms[6] and solution space smoothing into one process to search the optimal solution.

Algorithm: VP_SSS ()

STEP 1: create the initial placement instance according to the smoothing function.

STEP 2: use BSG_local_search to search the optimal solution for the initial placement instance. The result is a starting solution of the next smoothing instance.

STEP 3: α := f(α); apply the smoothing function to the previous solution to produce a new placement instance.

STEP 4: use BSG_local_search to search the solution for the new placement instance. The result is the current solution.

STEP 5: if α=1, stop. The current solution is the final solution. Otherwise, using the current solution as the next instance starting solution, go to STEP 3.

In VP_SSS( ), an initial value of α is chosen properly, in our experiments, we often set α_0 = 6. At the end of each iteration loop, α reduced by a function f(α). f(α) may simply equal to (α - 1), or it can be another decrease strategy to fine tune the shape of the smoothed solution space. For example, in our experiments we adopt the function f(α) as this:

f(α) = { (α - 1; when α > 2); (α - 0.20; when α > 1.5);
(α - 0.10; when α > 1.1); (α - 0.02; when α > 1)}

The time complexity of algorithm VP_SSS() can be estimated as follows. From a BSG assignment on BSG_{p*q} to its corresponding placement, it needs O(p*q) to fulfill the evaluation. Assume that in the BSG_local_search, *UNS* is the uncertain number of searches before the search process reach the solution, Ne is the number of searches used to ensure an optimal or nearly optimal solution was reached, Nn is the total number of the nets of the placement, so the local search process will need (O(*UNS* + Ne) * O(p*q)) +

O(Nn). Assume that each building block has an average number, say Pn, of pins, so the process to produce the initial placement instance will need O(n * Pn), this is the same as O(Nn). When f(α) = (α - 1), the loop will need α_0((O(*UNS* + Ne) * O(p*q)) + O(Nn)).

In our experiments, we set: Ne = 60000;

III. Experimental Results

We have implemented the algorithm VP_SSS( ) using C programming Language. For area and wire length optimization, the performance of our algorithm is superior to other algorithms proposed so far, the experimental results is not presented here because of page limitation. Here, We apply VP_SSS to placement with pre-placed modules and placement with considering congestion.

A. Placement with Pre-placed Modules

For placement with pre-placed modules[9], we test VP_SSS() as follows. If the calculated coordinates of the pre-placed module are less than the pre-placed coordinates assigned to the module, the pre-placed coordinates will be force-assigned to the pre-placed module. Otherwise, the calculated coordinates will be assigned to the pre-placed module. In object function, modules whose coordinates are more than pre-placed coordinate receive penalty without fail.

No loss of generality, we only use x-coordinate to illustrate our approach. The processing of y-coordinate is similar. The follow penalty item is added to the objective function: $\omega * \sum (x_i^a - x_i^p)$

In which ω is weight of penalty. x_i^a is the actual coordinate of a pre-placed module and x_i^p is the pre-placed coordinate of the module. The item calculates the differences between the actual coordinate and pre-placed coordinate of all pre-placed modules.

Fig.2 is a placement example of ami33 with four blocks pre-placed, its area is 1.25(mm^2), area usage is 92.47%, CPU time is 114 sec.. Fig.3 is a placement example of ami49 with 5 blocks pre-placed, its area is 37.40(mm^2), area usage is 94.76%, CPU time is 261 sec.. (on Sun Spark 20 station).

B. Placement with consideration of Congestion

As the technology advances, routability has become more and more important. So we must take into account the congestion factor, which is closely related to routability, during the placement stage. The congestion cost is often defined based on the rectangular global bins, into which the chip is divided. Assuming all the pins within a bin are located at the center of the bin, we can route all the nets along the global edges, which connect every pair of nearby centers of bins.

Given the routing supply S_e for each edge *e*, which is decided by technology parameters, size of bins, etc, we could use some model to estimate the routing demand D_e of that edge. The overflow of that edge equals to $D_e - S_e$, if it is congested, that is $D_e > S_e$, and zero otherwise, thus the total congestion is given by the summation of overflow for all the edges.

We adopt a probability model of routing estimation. Star model is used for every net. The probability model is used for each edge of the net. Suppose a net from *s* to *t* in Fig.1.

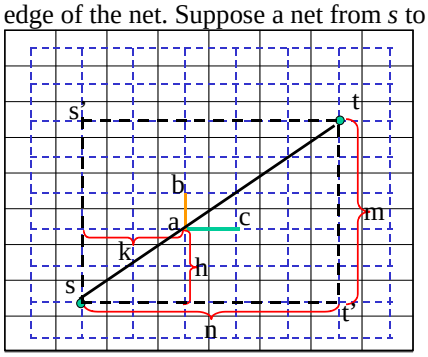

Fig.1 the route probability of a net

The net will probably route within the bound box of *ss'tt'*. The probability of the route along the edge *ac* and *ab*. are:

$$ac = \frac{\binom{h+k}{h}\binom{n-k-1+m-h}{m-h}}{\binom{m+n}{n}} \quad ab = \frac{\binom{h+k}{h}\binom{n-k+m-h-1}{m-h-1}}{\binom{m+n}{n}}$$

where the point *a* leaves *h* edges from *s* in vertical direction and *k* edges from *s* in horizontal direction. Then the route demand of every edge of the chip is gotten and the overflow edges are found.

Other than lots of previous efforts on congestion reduction, which optimize area, wirelength, and congestion step by step, we use an all-in-one cost function to deal with all these objectives together.

Some researchers[10] have observed that congestion cost, together with area and wirelength, is ill behaved, in another word, the associated solution space is quite rugged, and therefore it is hard to use it directly. However, as for our SSS algorithm, the rugged terrain could be smoothed gradually, thus the difficulty is overcome. The algorithm is basically the same as the VP_SSS stated before: we only add a congestion item to the equation:

$$f = area + \lambda * total\_wirelength + \mu * total\_congestion$$

where *total_congestion* is the summation of overflow of all the congested global edges.

Since different circuits have different wiring demands, and therefore different congestion situation, the weight of congestion factor μ should be different. To make the cost function sensitive to congestion without deteriorating area and wirelength too much, we recommend to choose μ that the production of μ and *total_congestion* is proximately 1~10 percents of area.

We list the experimental results in Table1 and compared its results with that of the algorithm, which only optimizes area and wirelength. We perform our tests on MCNC benchmarks, and the results are encouraging: in most of the cases, the total congestion has been significantly reduced. Fig.4 and Fig.5 are two examples before and after

considering congestion during placement .

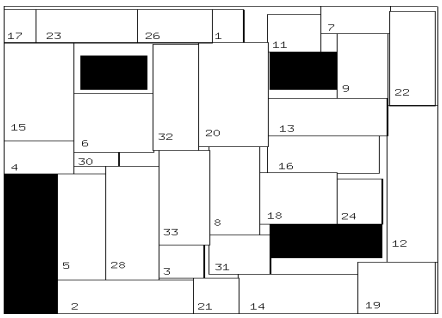

Fig.2 Ami33 placement with 4 pre-placed modules: the module number and its coordinates are: 10(700,800), 25(700,200), 27(0,0), 29(200,800).

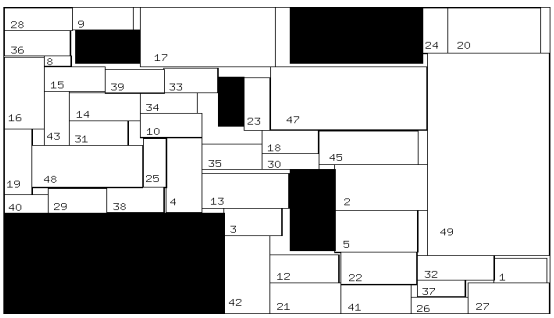

Fig.3 Ami49 placement with 5 pre-placed modules: 6(4000,1000), 7(1000,4000), 11(3000,3000), 46(0,0), 44(4000,4000)

Table1 Experimental results on VLSI module placement with considering congestion

	without considering congestion			with considering congestion			improvement
	Area	Wire	Congestion	Area	Wire	Congestion	Congestion
Xerox	20.59/21.32	564.3/710.9	140.9/288.7	20.42/21.35	424.9/660.6	96.2/154.8	31.7%/46.4%
Hp	9.34/9.84	153/175	7.06/38.5	9.40/10.1	149/196	0/5.53	100%/85.6%
Ami33	1.21/1.24	45.1/50.6	102.3/158.4	1.23/1.27	41.8/52.5	20.4/43.4	80.1%/66.9%
Ami49	37.25/37.65	936/1138	29.18/68.19	37.72/38.41	976/1123	0.0/2.70	100%/96.0%

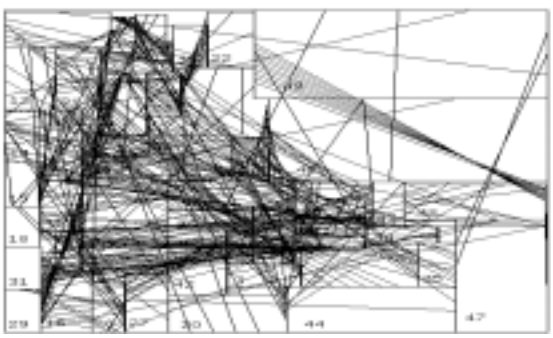

Fig.4 ami49 without considering congestion

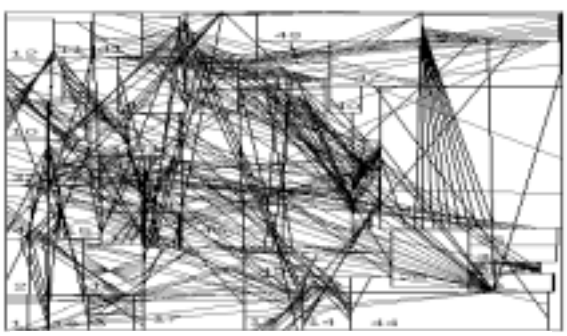

Fig.5 ami49 with considering congestion

IV. Conclusion

An effective solution space smoothing algorithm for VLSI module placement was designed in this paper. Compared with the placement algorithm based on simulated annealing, it uses only one parameter; say α_0, to control the search process. And α_0 is easy to determine for a placement instance, whereas, a cooling schedule of a simulated annealing process for a placement instance involve four parameters. Experimental results on placement with pre-placed modules and placement with considering congestion show that VLSI module placement algorithm based on solution space smoothing is an efficient and steadier placement algorithm. It is not common to consider congestion in BBL layout, the result shows the adaptability of Solution Space Smoothing algorithm.

References

[1] D.F.Wong, C.L. Liu, A new algorithm for floorplan design, Proc. of 23rd ACM/IEEE DAC'86, 101-107, 1986

[2] H.Murata, K.Fujyoshi, S.Nakatake, and Y.Kajitani, Rectangular-packing-based module placement, ICCAD'95, 472-479, 1995

[3] P. -N.Guo, C.-K.Cheng, T.Yoshimura, An O-tree representation of non-slicing floorplan and its applications , DAC'99, 1999

[4] S.Nakatake, K.Fujiyoshi, H.Murata, and Y.Kajitani, Module placement on BSG-structure and IC layout applications, ICCAD'96, 484-491, 1996

[5] Xianlong Hong, Gang Huang, Yici Cai, Jiangchun Gu, Sheqin Dong, C, -K Cheng, Jun Gu, Corner Block List: An Efficient and Effective Topological Representation of Non-Slicing Floorplan, ICCAD'00, 8-12.

[6] Shuo Zhou, Sheqin Dong, Xianlong Hong, Yici Cai, Chung-Kuan Cheng and Jun Gu, ECBL: An extended corner block list with solution space including optimum placement, ISPD'2001, 150-155.

[7] Jun Gu, Xiaofei Huang, Efficient Local Search with Search Space Smoothing: A Case Study of the Travelling Salesman Problem (TSP), IEEE Trans. On Systems. Man. and Cybernetics, Vol.24, No.5, 728-735, 1994

[8] Johannes Schneider, Markus Dankesreiter, Werner Fettes, Ingo Morgenstern, Martin Schmid, Johannes Maria Singer, Search-space smoothing for combinatorial optimization problems, Physica A 243(1997) 77-112.

[9] H.Murata, K.Fujiyoshi, M.Kaneko: "VLSI/PCB placement with obstacles based on sequence-pair, IEEE Trans. On CAD, Vol.17, No.1, pp60-68, 1998

[10]Maogang Wang, Majid Sarrafzadeh, Behavior of congestion minimization during placement. ISPD'99