ESE - 556 VLSI PHYSICAL AND LOGIC DESIGN AUTOMATION
Stony Brook University

PROJECT 2

DESIGN AND IMPLEMENT OF TIMBERWOLF SIMULATED ANNEALING PLACEMENT ALGORITHM

Submitted to
Professor Alex Doboli
Department of Electrical and Computer Engineering

Project Members:
Suchetha Panduranga - 111463372
Manoj Kumar Gopala - 111679371
Anup Mahadev - 111326655

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In floorplanning, our aim is to determine the relative locations of the blocks in the chip and the objective is to minimize the floorplan area, wirelength. Generally, there are so many strategies in VLSI floorplanning like area optimization, wirelength optimization, power optimization, temperature optimization and etc. This paper concentrates on area optimization. The goal of the physical design process is to design the VLSI chip with minimum area. The primary idea is to minimize the floorplan area by reshaping the blocks which are present inside the floorplan in order to attain the minimum area with less computational time. Proposed problem is redefined with an efficient meta-heuristic as Simulated Annealing algorithm which will provide optimal solution with less computation time.

INTRODUCTION

VLSI chip complexity has been increasing as per the Moore's law, demanding more functionality, high performance, but with less design time. Producing compact layouts with high performance in shorter time is required, in order to meet the time to market needs of today's VLSI chips. This calls for tools, which run faster and also which converge without leading to design iterations. Placement is the process of determining the locations of circuit devices on a die surface such that a specific objective function is optimized. It is the most important step in VLSI design flow as it affects the routability, total interconnect length, performance, time delay of critical nets and power consumption of the design. The main objective of this placement problem is to assign partitions to specific cells to achieve the cost function, subject to specific constraints. TimberWolf is an excellent placement algorithm developed by Sechen and Sangiovanni-Vincentelli that uses extensions and modifications of simulated annealing to carry out placement of the cells such that the total wire length is minimized, resulting in improved routability and reduced interconnect delay. It is deemed as an excellent heuristic for solving placement like combinatorial optimization problems. The basic procedure involved is to accept all moves that result in a reduction in cost. The program is designed to handle standard cell circuit configurations in which cells are arranged in horizontal rows in addition to allowing user-specified macroblocks and pads. The algorithm generally consists of rectilinear components being targeted as the rectangular or square die area in such a way that interconnect wire length is reduced. The application of simulated placement techniques is widely seen to make ultra-compact placements including power density and wiring congestion. The optimized placement technique enables the VLSI design to be logically and synthetically accurate as it adheres the given constraints. It is also involved in arranging the components inside the floorplan of the die or FPGA targeted for the construction of the integrated circuit implementing the original design.

This report presents the algorithm used by TimberWolf placement package and discusses the results that have been obtained. Section 2 discusses the relative work while Section 3 describes the problem

formulation and the associated implementation issues. Simulated annealing, the standard cell placement optimization algorithm is described in section 4. Section 5 discusses the results obtained for different IBM benchmarks after implementing the placement algorithm satisfying the given placement constraints. Section 7 is dedicated to remarks and conclusion of the topic.

PROBLEM STATEMENT

Determining a perfect placement solution is an NP-complete problem. For any design of considerable size, finding a complete solution requires overwhelming computational effort. Placement is a minimization problem meaning that one placement which has lower cost than another is more desirable and a placement which has the least cost is considered as the optimum

Given: *A set of n rectangular blocks* $i = 1,2 \dots n$
Circuit area A_i
Set of netlists (netlist) $N = \{N_1, N_2 \dots N_n\}$
Interconnect length L_i *for net* N_i
Timing of each building block T_i
Position of I/O pins on the pad frame

Goal: *Minimized total IC area* $A_{i(new)}$
Maximized routability of the interconnect $C_{ij(new)}$
Minimized interconnect length $L_{i(new)}$ *and Minimized time delay* $T_{i(new)}$ *of the critical net*

RELATED WORK

There are significant number of algorithms that addresses the NP complete dimension of placement problem. But one of the most elegant heuristic that applies to the VLSI cell placement problem is Simulated Annealing (SA). The placement process is compute intensive [1] [2] and represents a significant amount of time in the design flow. As with any of the steps in the design flow, it may be revisited several times to further optimize the design or to fix problems encountered by following steps. Placement acceleration would be desirable to any group implementing this step within a development path. Simulated Annealing [3] is a very common algorithm used to implement a placement tool [4] [5], accelerating this algorithm therefore has been in the center of many studies [6] [7]. Prior work in this field has firstly focused on improving the Simulated Annealing algorithm through analyzing and modifying the perturbation types and cost functions [8]. Other work has looked to parallel implementation of the Simulated Annealing algorithm purely in software to produce a speedup [9]. The serial nature of the algorithm does not directly lend itself to this approach though parallel implementation has been shown to be successful [10] [11]. Other approaches of speeding up the Simulated annealing algorithm have focused on hybrid implementations using other search methods as an augmentation [12] [13]. Hardware implementations have looked

towards specialized data and processing structures designed to be implemented in large matrices of execution elements on FPGAs [14].

Analyzing the above prior work, it seems a hardware accelerated parallel processing approach would have the ability to provide a substantial speedup. Applying knowledge of the behavior of the SA algorithm when applied to placement also provides novel approaches to a hardware accelerated approach. There are various other placement techniques like Breuer's algorithm [15], the placement is done based on the partition technique and the circuit is partitioned repeatedly into two sub circuits. [16] discusses the analytical approach to simulated annealing which is found to be the best approach as it minimizes the number of external connections between each ASIC and keeps each ASIC smaller than the maximum size. Much research has gone into parallelize the algorithm as discussed in [17] [18] and is widely applied to VLSI chip design problem.

BASIC SIMULATED ANNEALING ALGORITHM

Simulated annealing has been an effective method for the determination of global minima of combinatorial optimization problems involving many degrees of freedom. Its basic feature is the possibility of exploring the configuration space of the optimization problem allowing *hill climbing* moves. These moves are controlled by a parameter, in analogy with temperature in the annealing process, and are less and less likely towards the end of the process.

Given a combinatorial optimization problem specified by a finite set of configurations or states *S* and by a cost function *c* defined on all the states *j* in *S*, the simulated annealing algorithm is characterized by a rule to generate randomly a new state or configuration with a certain probability, and by a random acceptance rule according to which the new configuration is accepted or rejected. A parameter *T* controls the acceptance rule.

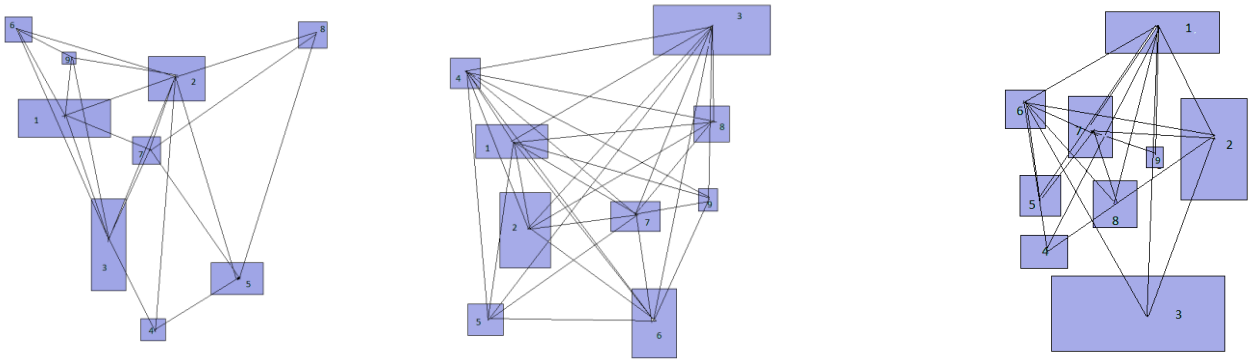
In the original version of simulated annealing, the acceptance function is governed by the function *f* shown below

$$f(\Delta c, T) = min[1, exp(- \Delta t/T)]. \qquad (1.0)$$

It is possible to vary the shape of function *f* by adjusting the control parameter *T,* called temperature. The updating rule for *T is* given as

$$Tnew = \alpha (T_{old}) * T_{old}, \quad 0< \alpha<1. \qquad (1.1)$$

In the function **accept,** note that new states characterized by $\Delta c \leq 0$ always satisfy the acceptance criterion. However, for the new states characterized by $\Delta c > 0$, the parameter *T* plays a fundamental role. If *T* is very large, then *r* is likely to be less than *y* and a new state is almost always accepted irrespective of Ac. If *T is* small, close to 0, then only new states that are characterized by very small $\Delta c > 0$ have any chance of being accepted. In general, all states with $\Delta c > 0$ have smaller chances of satisfying the test for smaller T.



(i) (ii) (iii)

Fig 1. Depiction of stages of simulated annealing (i) Before placement (ii) intermediate stage (iii) After placement)

THE TIMBER-WOLF IMPLEMENTATION OF THE SIMULATED ANNEALING ALGORITHM

Timber-Wolf optimizes the placement of standard cells into row and/or column blocks. Furthermore, the various blocks may have differing heights. The program also optimizes the placement of pads or buffer circuitry, as well as macro blocks. The macro blocks may be positioned anywhere on the chip. The estimation of the wire length for a single net is determined by computing the half-perimeter of the bounding box of the net. The bounding box is defined by the smallest rectangle which encloses all of the pins comprising the net. Because exact pin locations are used in the wire length calculations, Timber-Wolf considers all possible orientations for a cell, pad, or macro block. A group of pins which are internally connected within a cell must be given to Timber-Wolf as a single pin with a location which is the average of the locations of its constituent pins. The program employs the exchange class mechanism for blocks as well as cells, pads and macros. If two blocks have the same exchange class, then cells from these blocks are interchangeable. Blocks with differing exchange classes may not have their cells interchanged.

Differing exchange classes for blocks are usually employed when blocks have different heights. Furthermore, two cells or two pads may be interchanged only if they belong to the same exchange class. Timber-Wolf is applicable to standard cell placement problems of the complexity shown in Fig 1.

Cost Function: The cost function for the simulated annealing algorithm consists of following independent portions. The first portion is the total estimated wire length C1.

$$C1 = \sum_{net=1}^{numnets} ((xspan[net])(Hweight[net] + yspan[net])(Vweight[net]))$$

The second portion is the penalty function which consists of a total sum of overlap penalties. This penalty function was incorporated because of the usual difference in width of the standard cells. Often two cells are selected for interchange which differ in width Therefore, an exchange of location of these two cells often results in some overlap with one or more of the other cells.

$$C2 = \sum_{(i|=j)}((linearOverlap(i,j)) + constant)^2$$

The function linearOverlap returns the amount of overlap of the cells in the x direction. The offset parameter, which has a value of 3 in TimberWolf was chosen to ensure that when T approaches 0, then the value of C2 converges toward 0.

The third function, C3. is a penalty function which serves to carefully control the row lengths.

$$C3 = \sum_{net=1}^{numrows} |actRowLen[r] - desRowLen[r]| * parameter$$

where *actRowLen[r]* represents the sum of the widths of the cells currently placed in row r Furthermore, desRowLen[r] is the desired row length for row r .

The Generate function: The selection of new states is based on the following considerations: 1) A random number between one and the total number of cells, pads and macro blocks is generated. The cells are numbered from one to the total number of cells, and the pads and macro blocks are numbered starting from the number of cells plus one. If the random number is less than or equal to the number of cells, then a cell is selected. Otherwise, a pad or macro block is selected. 2) A second random number is selected between 1 and the total number of cells, pads, and macro blocks. 3) If the two numbers selected both represent cells, then the pair of cells are interchanged to generate a new state. 4) Similarly, if two pads or two macro blocks were selected, then an interchange constitutes the new state. 5) If the two numbers selected do not represent the same unit then the first unit selected governs the generation of a new state. If this first unit was a pad or macro block, then an orientation change of the respective unit is attempted. If the first unit was a cell, then this cell is displaced to a new location. If this new state is rejected, then the next state generated is an orientation change for the cell.

Range Limiter*:* The displacement of a cell to a new location is controlled by a *range limiter,* which limits the range of the displacement of a cell. As *T* is reduced, eventually the size of the range-limiter window has been reduced such that inter-block cell displacements or interchanges are no longer attempted. At this point, all residual cell overlaps are removed, and the blocks are compacted.

The generation of new states then takes on a different form as follows: 1) A standard cell is randomly selected and its left and right neighbors (if any) for the case of a row block or its bottom and top neighbors (if any) in the case of a column block are noted. 2) An interchange of the randomly selected cell is performed with either its left (bottom) neighbor and/or its right (top) neighbor for row (column) blocks. An orientation change of the selected cell is attempted if permitted by the user.

Generating New Values of T: In the current implementation of TimberWolf, the parameter *'α'* is user-specified as a versus *T* data. The best results have been obtained when α is the largest (approximately 0.95) during the stages of the algorithm when the cost function is decreasing rapidly. Furthermore, the value of α is given its lowest values at the initial and latter stages of the algorithm (usually 0.80).
The value of α is gradually increased from its lowest value to its highest value, and then gradually decreased back to its lowest value.

The Inner Loop Criterion: The inner loop criterion is implemented by the specification of the number of new states generated for each stage of the annealing process. This number is specified as a multiple of the number of fundamental units for the placement or routing problem. For the gate array placement and standard cell global router programs, 20 new states per unit are generated at each stage. The standard cell and macro/custom cell placement problems have many more degrees of freedom (orientation changes, pin location changes, etc.) and hence 100 or more new states are generated per cell at each stage.

The Stopping Criterion The stopping criterion is implemented by recording the cost function's value at the end of each stage of the annealing process. The stopping criterion is satisfied when the cost function's value has not changed for 3 consecutive stages.

TimberWolf is to insert route-through cells as necessary if the standard cell circuit contains only row blocks. A route-through cell must be inserted to accomplish this for the case of two levels of interconnect. Once the size of the range-limiter window has been reduced such that inter-block cell displacements or interchanges are no longer attempted, TimberWolf will then insert route-through cells as necessary. The route-through cells participate in the generation of new states as described above. That is, they are positioned in their respective rows such that the total wire length objective is minimized. For standard cell circuits comprised solely of row blocks of cells and pads around the periphery of the blocks, the TimberWolf will configure the rows in the most advantageous manner. The user inputs the number of rows desired and the estimated row separation, in anticipation of the fact that most of the route-through cells are concentrated toward the center-most rows, TimberWolf will restrict the total cell length allowable in these rows. The user supplies an indentfactor which is the ratio of the total cell length allowed in the center-most row divided by the total cell length allowed in the outermost row. The total cell length allowed in the other rows increases linearly from the center row toward each of the top and bottom rows.

TimberWolf also queries the user for the expected number of route-through cells. This can either be a guess or the user may try a short TimberWolf run and note the number required. TimberWolf uses this information to increase the actual row lengths. Note that when the final placement is determined by TimberWolf and the route-through cells have been added, the final row lengths will tend to be close to the actual row lengths given to TimberWolf. Having the actual row lengths greater than the total allowable cell length for each row increases the cardinality of the state space of the problem and yields the best results.

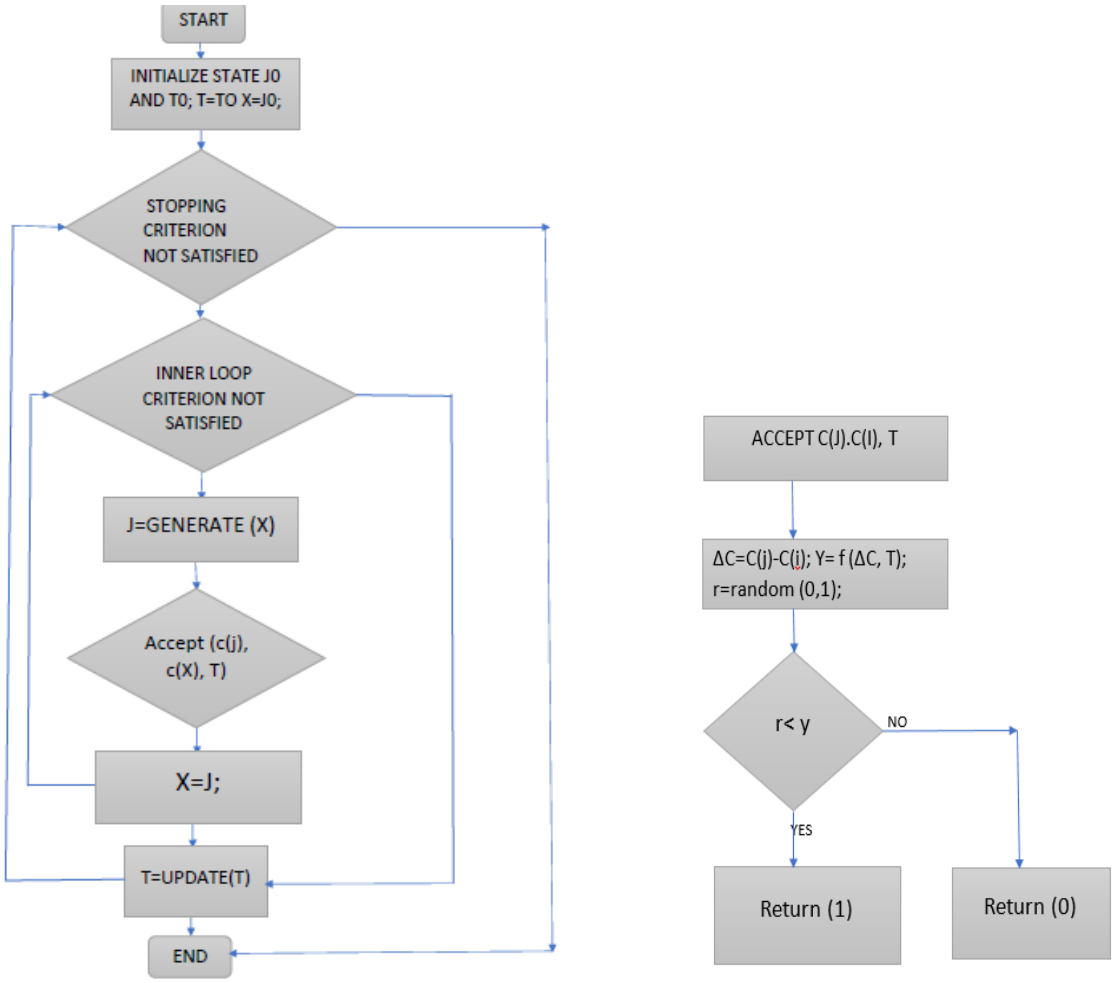


Fig 4. Flowchart for the control flow - Simulated algorithm

IMPLEMENTATION ISSUES

Mapping a real problem to the domain of simulated annealing can be difficult and requires familiarity with the algorithm. More often than not, it is possible to arrive at the solution using several approaches. In addition, there are other factors that determine the success or failure of this algorithm. Several practical considerations for the proper implementation of simulated annealing include how to perturb the solution, how to decide a proper cooling schedule. The initial temperature is too high which will result in the cost function going higher than the initial value. Thus, applying too much perturbation is redundant. This raises the questions like how long the perturbation should be applied and how much is to be applied to keep the cost function from oscillating too vaguely.

At the beginning of the process, the temperature and error are high and as time goes by, the temperature decreases slowly, reducing the error function. However, it is possible that the process completes without finding an optimal solution. The parameters for simulated annealing should be carefully selected that will reduce the probability of this to happen. Due to the large size of the layout and the original

layout, the Timber-Wolf algorithm cannot significantly reduce the line length and overlap in a limited amount of time.

EVALUATION AND RESULTS

The C++ Source code was run on Windows 10 Operating System with the system configuration of 16GB RAM and Intel I7 processor. We were able to run *15 IBM benchmarks* successfully. This section presents the algorithm's performance with respect to various annealing parameters. We observed that the final wire length was reduced by 2-3% for different benchmarks. The overlap measure that determines the delay in interconnects was reduced as well. This resulted in considerable increase in routability between the nets. To explain the change in wirelength with respect to temperature, we consider benchmarks IBM01 and IBM05.

Evaluation of the algorithm against benchmarks

IBM 1

The variation of wirelength as a function of temperature was plotted and the wirelength was found to decrease linearly until the temperature reaches 25000 and then start fluctuating between occasional uphill jumps, allowing them to hop out of local minima and giving them a better chance of finding the global minimum.

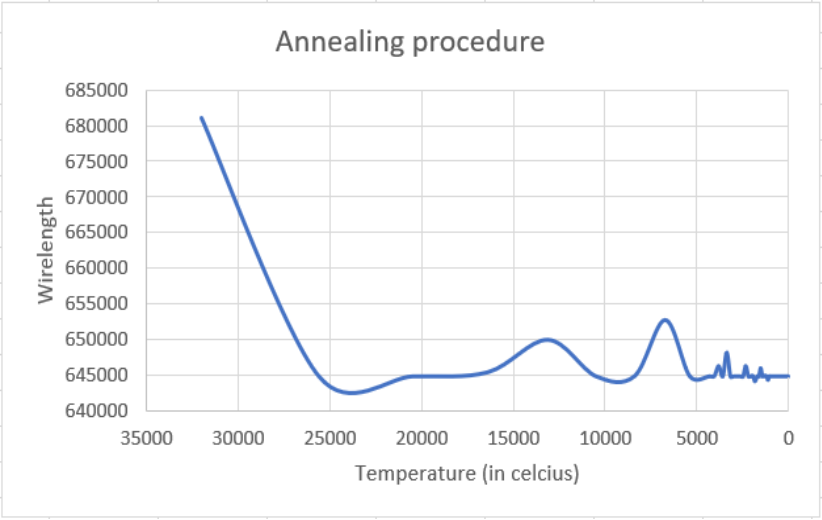


Fig 3 (a): Wirelength vs Temperature for IBM01

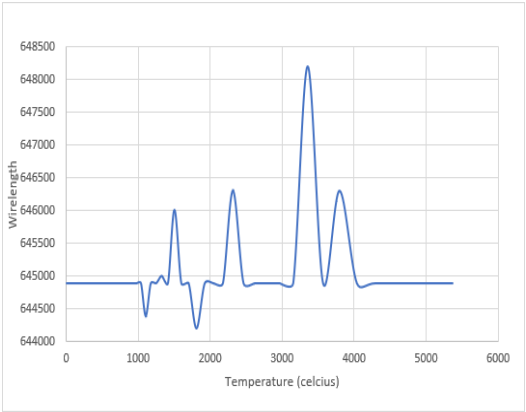


Fig 3 (b): Uphill jumps, hop out of local minima

Cell Placement for IBM01: Cell Placement was plotted using MATLAB R2017b. We examined how the cells were placed before and after the Timberwolf Algorithm execution. The inputs to the construction were (X, Y) co-ordinates and (L,W) of the cells**.**

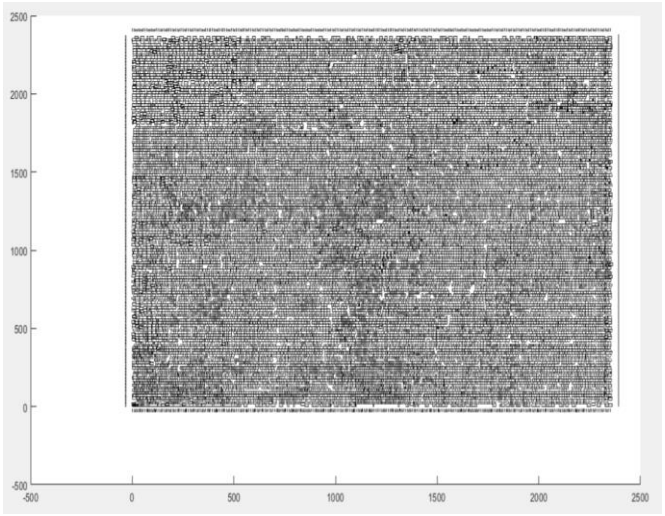


Fig 4 (a): Before placement for IBM01

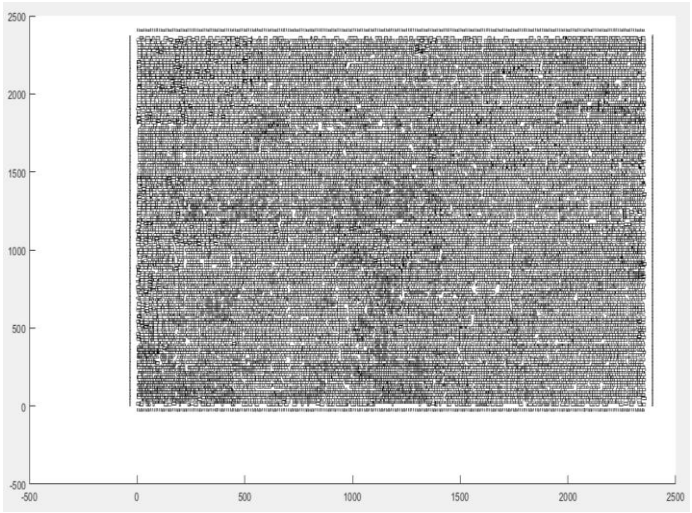


Fig 4(b): After placement for IBM01

The plot shows the no of iterations as a function of the control parameter α

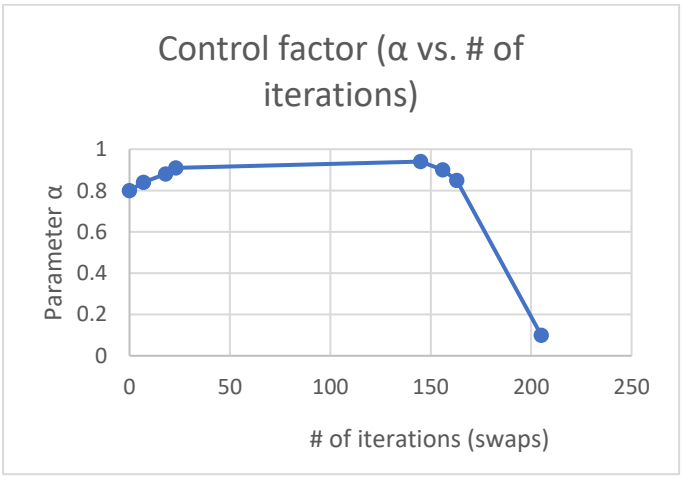
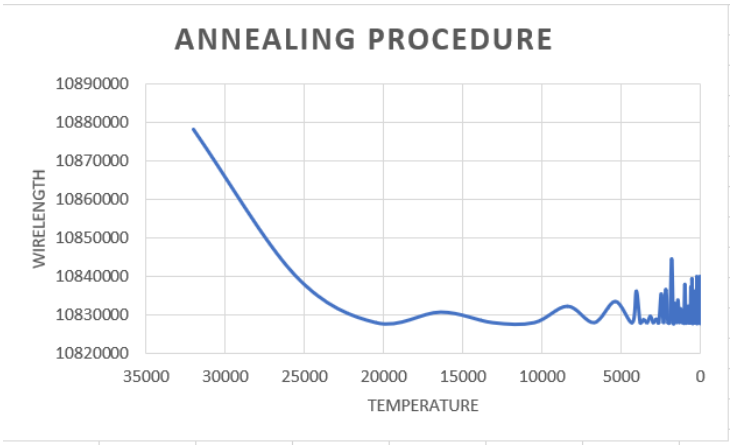


Fig 4(c) Control factor α vs. iterations

IBM 05



As we can see from the graph the wirelength decreases initially linearly until 20000 and later the wirelength kept fluctuating between occasional uphill jumps, allowing them to hop out of local minima and giving them a better chance of finding the global minimum.

Fig 5: Wirelength vs Temperature for IBM05

Cell Placement for IBM05:

Cell Placement was plotted using the MATLAB R2017. We examined how the cells were placed before and after the Timberwolf Algorithm execution. The inputs to the construction were (X, Y) co-ordinates and (L,W) of the cells

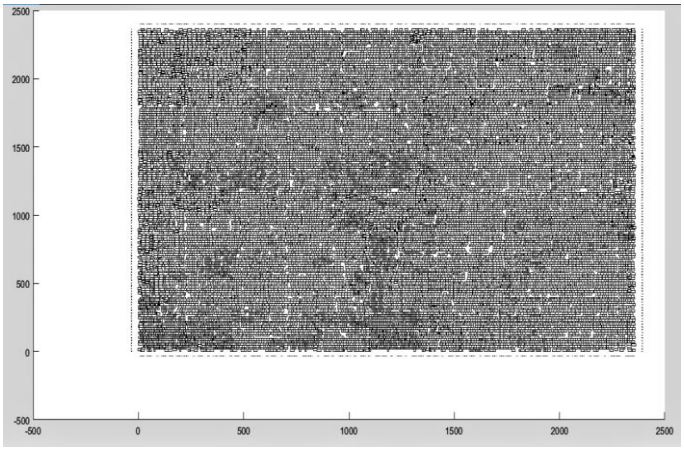


Fig 6 (a): before placement for IBM05

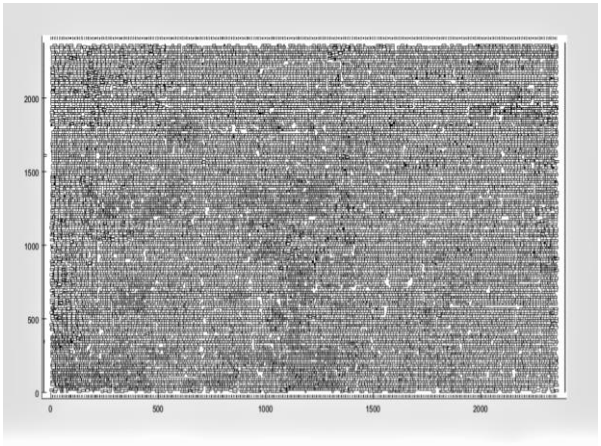
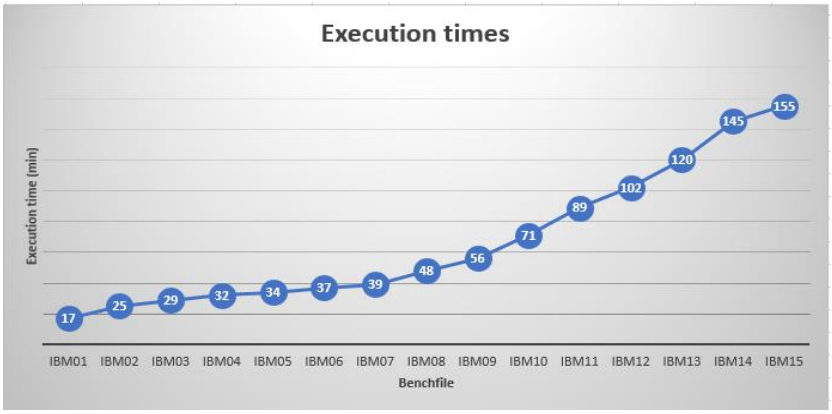


Fig 6 (b): After placement for IBM05

Execution Times for IBM files



The execution times for each benchmark is plotted and it is found to increase linearly with increase in the number of cells in the benchmark files

Fig 7: execution times for IBM files

Final Wirelength and execution ties of IBM files

Bench	#nets	#cells	Intial wirelength	Final wirelength	Execution time (Mins)
ibm01	246	12506	698900	646701	17
ibm02	259	19342	1201456	10009562	25
ibm03	283	22853	2104030	1875432	29
ibm04	287	27220	2300214	1983240	32
ibm05	1201	28146	10843314	10780324	34
ibm06	166	32332	984223	969943	37
ibm07	287	45639	2734100	2524540	39
ibm08	286	51023	3102345	2906995	42
ibm09	285	53110	3367000	2987540	44
ibm10	744	68685	8932142	7633442	71
ibm11	406	70152	4523103	4255163	89
ibm12	637	70439	9392000	9102440	102
ibm13	490	83709	5723901	5443954	120
ibm14	517	147088	4982312	4822312	143
ibm15	383	161187	7068595	6676120	155

Table 1: Result statistics for IBM files

The execution times and final wirelength after the implementation of the algorithm was evaluated against 15 IBM benchmark files and results were tabulated. It was found that the final wirelength was reduced by 2-3%

CONCLUSION

This work began by implementing and characterizing the Simulated Annealing algorithm with respect to component placement and optimization inside of a software program. This step allowed insight to be gained into the intricacies of the SA algorithm giving the ability to conceive a novel approach to modeling an optimized placement solution. Having characterized and understood the serial SA algorithm, efforts were made to arrive at the best possible solutions for cell placement that would result in a minimized interconnect delay and take the routability a notch up. As a future work, we aim to study other algorithms like Genetic algorithm and Particle Swarm Optimization algorithm to solve this modern VLSI floor planning problem.

BIBILOGRAPHY

1. S. Sait and H. Youssef VLSI Physical Design Automation. World Scientific Publish ing Company, 1999.
2. N. Sherwani. Algorithmsfor Physical Design Automation. Kluwer Academic Pub lishers, Boston, MA, 1992
3. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, 220, 4598:671-680, 1983.
4. S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, and I. L. Markov. Unification of partitioning, floorplanning and placement. In International Conference ofComputer AidedDesign (ICCAD), San Jose, 2004
5. C. Sechen and A. Sangiovanni-Vincentelli. The TimberWolf placement and routing package. IEEE Journal ofSolid-State Circuits, SC-20(2), Apr 1985.
6. C. Sechen. VLSI Placement andGlobalRouting Using SimulatedAnnealing. Kluwer Academic Publishers, Boston, MA, 1988
7. R. Tessier. Frontier: A fast placement system for FPGAs. In Proceedings ofthe IFIP TC10/WG10.5 Tenth International Conference on Very Large Scale Integration, pages 125-136, 2000.
8. J. A. Chandy, S. Kim, B. Ramkumar, S. Parkes, and P. Banerjee. An evaluation of parallel simulated annealing strategies with application to standard cell placement. In Trans, on Computer Aided Design ofIntegrated Circuits andSystems, Apr 1997.
9. M. D. Durand and S. R. White. Trading accuracy for speed in parallel simulated annealing with simultaneous moves. Parallel Computing, 26(1): 135-150, 2000
10. R. F. Hentschke and R. A. da Luz Reis. Improving simulated annealing placement by applying random and greedy mixed perturbations. In Proc. ofthe 16th Symposium on Integrated Circuits andSystems Design, 2003.
11. S. A. Kravitz and R. A. Rutnebar. Placement by simulated annealing on a multipro cessor. IEEE Transactions on Computer-AidedDesign, CAD-6(4):534-549, 1987.
12. M. D. Durand and S. R. White. Trading accuracy for speed in parallel simulated annealing with simultaneous moves. Parallel Computing, 26(1): 135-150, 2000
13. R. F. Hentschke and R. A. da Luz Reis. Improving simulated annealing placement by applying random and greedy mixed perturbations. In Proc. ofthe 16th Symposium on Integrated Circuits andSystems Design, 2003.
14. J. A. Khan and S.M. Sait. Fuzzy aggregating functions for multiobjective VLSI place ment. In Proceedings ofthe 2002 IEEE International Conference on Fuzzy Systems, volume 2, pages 831-836, 2002
15. http://limsk.ece.gatech.edu/book/papers/breuer.pdf
16. Analog circuit design optimization based on symbolic simulation and simulated annealing. IEEE Journal of Solid-State Circuits (Volume: 25, Issue: 3, Jun 1990)
17. https://www.cs.montana.edu/~atanu.roy/CS545/Simulated%20Annealing%20Project%20Report.pdf, Atanu Roy Karthik and Ganesan Pillai. Parallel Simulated Annealing for VLSI Cell Placement Problem. Department of Computer Science, Montana State University – Bozeman
18. A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

APPENDIX

Functions	Description
inputnodes()	Reads length, width, and retention information for each module from .node file
Weightinput()	Reads weights from the .wts file from the bookshelf format
Plinput()	Reads the location and orientation of each module from the .pl file
Netsinput()	Reads the Wired Network Information from .net file
Sclinput()	Reads the circuit information for each line of information from .scl file
gnuPlot()	create a human readable format of the cell coordinates along with size of output.txt format
boundcalc()	calculate boundaries of the cell
wireLength()	calculate the wirelength offset
cellOverlap()	Calculates the total overlap
initiateMove()	To check the validly to swap the cells
update_Temperature()	Calculates the α and update the temperature
timberWolf()	will initiate the cell move and temperature update.

Table 2. Functions employed for software implementation and their definitions

Copy of the source code

The software implementation has two parts, a source code containing the function definitions and the main trigger function to initiate moves and implement the algorithm to find the optimal solution in an iterative process. The other file being the header file has the function declarations

Main.cpp

```
#include<iostream>
#include<vector>
#include<fstream>
#include<boost/algorithm/string/split.hpp>
#include<boost/algorithm/string.hpp>
#include<string>
#include <iostream>
#include <sstream>
#include <regex>
#include <map>
#include <cmath>
#include "time.h"
#include "main_head.h"

void inputnodes()
  {
        fstream file;
        string buf;
        int i=0;
        vector<string> strVec;
        using boost::is_any_of;
        int Value=2;

        file.open("ibm03.nodes", ios :: in);    //open input file - nodes
        while (getline(file, buf))
        {
              i++;
              if(i>7)
              {
                    boost::algorithm::split(strVec,buf,is_any_of("
,\t"),boost::token_compress_on);
                    node n;
                    //cout<<strVec.size()<<endl;
                    if (strVec[4] == "terminal")
                    {
                          Value = 1;
                    }
                    else
                    {
                          Value = 0;
                    }
                    //cout<<strVec[1]<<"                              "<<atoi(strVec[2].c_str())<<"
"<<atoi(strVec[3].c_str())<<""<<Value<<endl;;
```

```
        n.setParameterNodes(strVec[1],atoi(strVec[2].c_str()),atoi(strVec[3].c_str()),Value);
                                nodeId.insert(pair<string,node>(strVec[1],n));
                        }
                }
                file.close();
          }

        void weightinput()        //weights fromthe wts file from the bookshelf format
          {
                fstream file;
                string buf;
                int i=0;
                vector<string> strVec;
                using boost::is_any_of;
                map<string, node>::iterator itr;

                file.open("ibm03.wts", ios :: in);

                while (getline(file, buf))
                {
                        i++;
                        if(i>5)
                        {
                                boost::algorithm::split(strVec,buf,is_any_of("
,\t"),boost::token_compress_on);
                                nodeId[strVec[1]].setParameterWts(atoi(strVec[2].c_str()));

                        }
                }
                file.close();
          }

        void Plinput()      //Pl input file from the benchmark group
          {
                fstream file;
                string buf;
                int i=0;
                vector<string> strVec;
                using boost::is_any_of;

                file.open("ibm03.pl", ios :: in);

                while (getline(file, buf))
                {
```

```
                        i++;
                        if(i>6)
                        {
                                boost::algorithm::split(strVec,buf,is_any_of("
,\t"),boost::token_compress_on);

        nodeId[strVec[0]].setParameterPl(atoi(strVec[1].c_str()),atoi(strVec[2].c_str()),strVec[4]);

                        }
                }
                file.close();
          }

        void Netsinput()     //netlist from the benchmark group
        {
                fstream file;
                string buf;
                int i=0,a=0,j=0,NetId=1;
                vector<string> strVec;
                using boost::is_any_of;

                regex find ("\\b(NetDegree : )");
                smatch match;
                string Out;

                file.open("ibm03.nets", ios :: in);

                while (getline(file, buf))
                {
                        i++;
                        if(i>7)
                        {
                                regex_search(buf, match, find);
                                Out = match.suffix();
                                a = atoi(Out.c_str());
                                vector<string> strTemp;
                                for (j=0; j<a; j++)
                                {

                                        getline(file,buf);
                                        boost::algorithm::split(strVec,buf,is_any_of("
,\t"),boost::token_compress_on);
                                        strTemp.push_back(strVec[1]);
                                        nodeId[strVec[1]].setNetList(NetId);
                                }
```

```
                        netToCell.insert(pair<int, vector <string> >(NetId,strTemp));
                        NetId++;
                }

        }
}

void plotmap()
{
        map<int, vector<string> > :: iterator itr;
        vector<string> :: iterator itr1;

        cout << "netToCellMap" << endl;
        if(itr != netToCell.end()){
        for(itr= netToCell.begin(); itr != netToCell.end(); ++itr)
        {
                cout << itr->first <<"   ";
                if(itr1 != itr->second.end()){
                for(itr1 = itr->second.begin(); itr1 != itr->second.end(); ++itr1 )
                {
                        cout << *itr1 <<"        ";
                }}

                cout << endl;
        }}
        cout << "\n" << endl;
}

void Sclinput()      //.scl file from the ibm benchmark
  {
        fstream file;
        string buf;
        int i=0,j=0,Id=1;
        vector<string> strVec;
        using boost::is_any_of;

        int coOrdinate;
        int height;
        int siteWidth;
        int siteSpacing;
        string siteOrient;
        string siteSymmetry;
        int siteRowOrigin;
        int numSites;
```

```
		file.open("ibm03.scl", ios :: in);

		while (getline(file, buf))
		{
			i++;
			if(i>8)
			{
				boost::algorithm::split(strVec,buf,is_any_of("
,\t"),boost::token_compress_on);
				j = i%9;
				if(j == 1)
				{
					coOrdinate = atoi(strVec[3].c_str());
				}
				else if(j == 2)
				{
					height = atoi(strVec[3].c_str());
				}
				else if(j == 3)
				{
					siteWidth = atoi(strVec[3].c_str());
				}
				else if(j == 4)
				{
					siteSpacing = atoi(strVec[3].c_str());
				}
				else if(j == 5)
				{
					siteOrient = strVec[3];
				}
				else if(j == 6)
				{
					siteSymmetry = strVec[3];
				}
				else if(j == 7)
				{
					siteRowOrigin = atoi(strVec[3].c_str());
					numSites = atoi(strVec[6].c_str());
				}
				else if(j == 8)
				{
					row r;
					r.setId(Id);
					rowId.insert(pair<int,row>(Id,r));
```

```
        rowId[Id].setParameterRows(coOrdinate,height,siteWidth,siteSpacing,siteOrient,siteSymmetry,sit
eRowOrigin,numSites);
                                        Id++;
                                }

                        }
                }
                file.close();
          }

        void gnuPlot( string fileName )   //create a human readable format of output
        {
        ofstream myfile (fileName);
                if (myfile.is_open())
                {
                        int x = 0,y = 0,w = 0,h = 0;
                        myfile << "x-cord" << "   " << "y-cord" << "   " << "width" << "   " << "height" << endl;

                        for(map<string,node>::iterator      it_node      =      nodeId.begin();it_node      !=
nodeId.end();++it_node)
                        {

                                x = it_node->second.xCoordinate;
                                y = it_node->second.yCoordinate;
                                w = it_node->second.width;
                                h = it_node->second.height;
                                        /*float t = Temperature;
                                        long int wl = wireLength;*/

                                        myfile << x << "     " << y << "     " << w << "     " << h << "     " << endl;
                        }


                        //myfile.close();
                }
                else cout << "Unable to open file";
        }


        struct boundaries
        {
        int minX, maxX, minY, maxY;
        };

        int  numNodes =0;
```

```
float numCellsPerRow = 0;
boundaries b;

void boundcalc()     //calculate boundaries of the cell
{
int xval,yval;
b.minX = 32767, b.maxX = -32768,b.minY = 32767, b.maxY = -32768;
        map<string, node> ::iterator itNode;
        for(itNode = nodeId.begin(); itNode != nodeId.end(); ++itNode)
        {
                if(itNode->second.terminal == 1)
                {
                        xval = itNode->second.xCoordinate;
                        yval = itNode->second.yCoordinate;
                        if(xval < b.minX)
                        {
                                b.minX = xval;
                        }
                        if(xval > b.maxX)
                        {
                                b.maxX = xval;
                        }
                        if(xval < b.minY)
                        {
                                b.minY = yval;
                        }
                        if(xval > b.maxY)
                        {
                                b.maxY = yval;
                        }

                }
        }
        if (b.minX==32767)
                cout<<"error"<<endl;
                b.minX = 0;
        if (b.minY == 32767)
                b.minY = 0;
}

void convertrowtocell()
{
        map<int, row> ::iterator itRow;
        map<string, node> ::iterator itNode;
        for(itRow = rowId.begin(); itRow != rowId.end(); ++itRow)
```

```
		{
			for(itNode = nodeId.begin(); itNode != nodeId.end(); ++itNode)
			{
				if(itNode->second.height = 16)
				{
				if((itRow->second.coOrdinate <= itNode->second.yCoordinate) && ((itRow->second.coOrdinate)+(itRow->second.height)) >= itNode->second.yCoordinate)
				{
					itNode->second.setRowId(itRow->first);
					itRow->second.setCellList(itNode->first);
				}
				}
			}
		}
	}

	//void printEverything();
	void placeintial();
	void showcell();
	void showrow();
	int calcoverlaprow();
	void updateCenter();
	long int wireLength();
	int macroPlacement();
	long int cellOverlap();
	void timberWolf();
	void timberWolf2();
	void update_Temperature();
	void initiateMove(int xLimit);
	bool checkMove(long int prevCost);

	double Temperature;

	int main()
	{
		time_t timeStart = time(NULL);
		cout << "Time counter set to 0" << endl;

		inputnodes();
		weightinput();
		Plinput();
		Netsinput();
		Sclinput();

		updateCenter();
```

```
        convertrowtocell();
        cout << "Reading file " << "ibm03" << endl;
        cout << "Initial wirelength" << wireLength();
        cout << endl;
        gnuPlot("before1.txt");
        timberWolf();
        gnuPlot("after1.txt");
        timberWolf();
        gnuPlot("after2.txt");
        timeStart = time(NULL) - timeStart;
        float a = timeStart;
        cout << "Execution Time: " << (double)timeStart / 60.0 << endl;
        cout << "Final wirelength " << wireLength();
}



int macroPlacement()
{
        int xValue=b.minX,yValue=0;
        xLimit=b.minX;
        map<int, row> ::iterator itRow;
        itRow = rowId.begin();
        int rowHeight = itRow->second.height;

        map<string,node>::iterator itNode;
        for(itNode = nodeId.begin();itNode != nodeId.end();++itNode)
        {
                if(itNode->second.terminal == false && itNode->second.height > rowHeight)
                {
                        if(xValue + itNode->second.width > xLimit)
                        {
                                xLimit = xValue + itNode->second.width+1;
                        }
                        if(yValue + itNode->second.height < b.maxY)
                        {
                                itNode->second.yCoordinate = yValue;
                                itNode->second.xCoordinate = xValue;
                        }
                        else
                        {
                                yValue = 0;
                                xValue = xLimit;
                                itNode->second.yCoordinate = yValue;
```

```
                                itNode->second.xCoordinate = xValue;
                        }
                        yValue = yValue + itNode->second.height;
                }
        }
        return xLimit;
}

void placeinitial()
{
map<string, node> ::iterator itNode;
map<int, row> ::iterator itRow=rowId.begin();
int totalWidth = 0, rowWidth = 0, cnt=0,count=0, xCord=xLimit, yCord=0;
for(itNode = nodeId.begin(); itNode != nodeId.end(); ++itNode)
{
        if(itNode->second.terminal == 0 && itNode->second.height == 16)
        {
                totalWidth += itNode->second.width;
                numNodes++;
        }
}
itNode = nodeId.begin();


totalWidth = totalWidth + numNodes;
rowWidth = totalWidth/rowId.size();
numCellsPerRow = ceil(float (numNodes)/rowId.size())-1;


cout<<"rowNum="<<rowId.size()<<endl;
cout<<"nodeNum(not fixed)="<<numNodes<<endl;

}

int calcoverlaprow()
{
        map<int, row> ::iterator itRow;
        int cost3=0;

        for(itRow = rowId.begin(); itRow != rowId.end(); ++itRow)
        {
                itRow->second.calcRowOverlap();
                if(itRow->second.overlap >= 0)
                {
                        cost3 += itRow->second.overlap;
```

```
                }
        }
        cout<<"Overlap : "<<cost3<<endl;
return cost3;
}

void updateCenter()
 {
        map<string, node> ::iterator itNode;

        int xBy2 =0, yBy2 =0;
        for(itNode = nodeId.begin(); itNode != nodeId.end(); ++itNode)
        {
                xBy2 = (itNode->second.xCoordinate)+((itNode->second.width)/2);
                yBy2 = (itNode->second.yCoordinate)+((itNode->second.height)/2);
                itNode->second.setCenter(xBy2,yBy2);
        }
 }

long int wireLength() //calculate the wirelength offset
{
        map<int, vector<string>> :: iterator itNet;
        vector<string> :: iterator itCellList;
        int xVal, yVal, wireLength=0;
        //cout<< "updating wire length "<< endl;
        for(itNet= netToCell.begin(); itNet != netToCell.end(); ++itNet)
        {
                        int minXW = 32767, minYW = 32767, maxXW = -32768 , maxYW = -32768;
                        for(itCellList = itNet->second.begin(); itCellList != itNet->second.end();
++itCellList)
                        {
                                //cout<<"i am in";
                                xVal = nodeId[*itCellList].xBy2;
                                yVal = nodeId[*itCellList].yBy2;
                                //cout<<"x = "<<xVal<<" y = "<<yVal<<endl;
                                if(xVal < minXW)
                                minXW = xVal;
                                if(xVal > maxXW)
                                maxXW = xVal;
                                if(yVal < minYW)
                                minYW = yVal;
                                if(yVal > maxYW)
                                maxYW = yVal;
                        }
                        //cout<<"maxXW "<<maxXW<<" minXW "<<minXW<<" maxYW
```

```
"<<maxYW<<"  minYW   "<<minYW<<endl;
                        wireLength += abs((maxXW-minXW)) + abs((maxYW-minYW));
                }
                        cout << "Updated Wire length after move: " << wireLength << endl;
        return wireLength;
        }

        long int cellOverlap()
        {

        map<string,node>::iterator nodeit=nodeId.begin();
        map<int,row>::iterator rowit;
        int x1=0,w1=0,x2=0,w2=0,h1=0,h2=0,overlap=0,i=0;
        long int totaloverlap=0;

        vector<string>list;
        for(rowit=rowId.begin();rowit!=rowId.end();++rowit)
        {
                overlap=0;
                list=rowit->second.sortByX();
                overlap=0;
                for(i=0;i<list.size();i++)
                {
                        node nodeobj=nodeId.find(list[i])->second;
                        x1=nodeobj.xCoordinate;
                        w1=nodeobj.width;
                        h1=nodeobj.height;
                        i++;
                        if(i==list.size())
                        {
                                break;
                        }
                        x2=nodeId.find(list[i])->second.xCoordinate;
                        w2=nodeId.find(list[i])->second.width;
                        h2=nodeId.find(list[i])->second.height;
                        h1 = h2>h1?h1:h2;
                        if(x2<=(x1+w1)&&(x2+w2)>=(x1+w1))
                        {
                                overlap+=((x1+w1)-x2)*h1;
                        }
                        else if(x2>=x1 && x2<=(x1+w1) && (x2+w2)<=(x1+w1))//原来为(x2+w2)<=w1
                        {
                                overlap+=w2*h1;
                        }
                        i--;
```

```
        }
totaloverlap+=overlap;

}
cout << "Total cell overlap :" << totaloverlap << endl;
return totaloverlap;
}

void initiateMove(int xLimit)
{
srand(time(NULL));
int randomCellnum, randRow, randXcord;
stringstream randomCellTemperature;
string randomCellStr,randomCell, randomCell2;
node n;
row r;
map<string, node> ::iterator itNode, itNode2;
map<int, row> ::iterator itRow, itRow2;
bool accept;
long int cost2,cost1,prevCost;
//int cost3;
        //cost3 = rowOverlap();
        cost1 = wireLength();
        cost2 = cellOverlap();
        prevCost = cost1 + cost2;

        //cout<<numNodes<<endl;
        randomCellnum = rand() %  numNodes +1;
        randomCellTemperature << randomCellnum;
        randomCellStr = randomCellTemperature.str();
        randomCell = "a" + randomCellStr;

if (rand()/(RAND_MAX+0.0)<=0.8)
{
        randRow = rand() % (rowId.size()) + 1;
        randXcord = rand() %((int)numCellsPerRow)+xLimit;

        itRow = rowId.find(randRow);
        itNode = nodeId.find(randomCell);
        n = itNode->second;
        //r = itRow->second;
        cout<<" moving cell "<<randomCell<<" to Row "<<randRow<< endl;
        itNode->second.yCoordinate = itRow->second.coOrdinate;
        itNode->second.xCoordinate = randXcord;
```

```
                rowId[itNode->second.cellRowId].cellList.erase(std::remove(rowId[itNode-
>second.cellRowId].cellList.begin(),          rowId[itNode->second.cellRowId].cellList.end(),randomCell),
rowId[itNode->second.cellRowId].cellList.end());
                itRow->second.cellList.push_back(randomCell);
                itNode->second.cellRowId = randRow;
                updateCenter();

                accept = checkMove(prevCost);
                //accept = 0;
                if(!accept)
                {
                        itNode->second = n;
                        itRow->second.cellList.erase(std::remove(itRow->second.cellList.begin(),itRow-
>second.cellList.end(),randomCell), itRow->second.cellList.end());
                        rowId[itNode->second.cellRowId].cellList.push_back(randomCell);
                        if (rand()/(RAND_MAX+0.0)<=0.1)
                        {
                                itNode->second.setOrientation(intToOrient(rand()%4));
                                accept = checkMove(prevCost);
                                if(!accept)
                                        itNode->second = n;
                        }
                }
        }
        else
        {
                randomCellnum = rand() %  numNodes +1 ;
                randomCellTemperature.str("");
                randomCellTemperature << randomCellnum;
                randomCellStr = randomCellTemperature.str();
                randomCell2 = "a" + randomCellStr;

                cout<<"exchange "<<randomCell<<" with "<<randomCell2<< endl;

                int n1, n2;
                node nodeSec1, nodeSec2;
                itNode = nodeId.find(randomCell);
                itNode2 = nodeId.find(randomCell2);
                n1 = itNode->second.cellRowId; n2 = itNode2->second.cellRowId;
                //cout<<"originally, n1 = "<<n1<<" n2 = "<<n2<<endl;
                itRow = rowId.find(n1);
                itRow2 = rowId.find(n2);
                nodeSec1 = itNode->second;
                nodeSec2 = itNode2->second;
```

```
		int xTemp, yTemp;
		xTemp = itNode->second.xCoordinate;
		yTemp = itNode->second.yCoordinate;
		itNode->second.xCoordinate = itNode2->second.xCoordinate;
		itNode->second.yCoordinate = itNode2->second.yCoordinate;
		itNode2->second.xCoordinate = xTemp;
		itNode2->second.yCoordinate = yTemp;

		//itRow->second.cellList.erase(std::remove(rowId[itNode-
>second.cellRowId].cellList.begin(), rowId[itNode->second.cellRowId].cellList.end(),randomCell),
rowId[itNode->second.cellRowId].cellList.end());
		itRow->second.cellList.erase(std::remove(itRow->second.cellList.begin(), itRow-
>second.cellList.end(),randomCell), itRow->second.cellList.end());
		itRow->second.cellList.push_back(randomCell2);
		itNode->second.cellRowId = n2;

		//itRow2->second.cellList.erase(std::remove(rowId[itNode2-
>second.cellRowId].cellList.begin(), rowId[itNode2->second.cellRowId].cellList.end(),randomCell2),
rowId[itNode2->second.cellRowId].cellList.end());
		itRow2->second.cellList.erase(std::remove(itRow2->second.cellList.begin(), itRow2-
>second.cellList.end(),randomCell2), itRow2->second.cellList.end());
		itRow2->second.cellList.push_back(randomCell);
		itNode2->second.cellRowId = n1;

		updateCenter();

		accept = checkMove(prevCost);
		//accept = 0; //i set it to 1 to debug
		if(!accept)
		{
			n1 = itNode->second.cellRowId; n2 = itNode2->second.cellRowId;
			//cout<<"after exchange, n1 = "<<n1<<" n2 = "<<n2<<endl;
			itNode->second = nodeSec1;
			itRow->second.cellList.erase(std::remove(itRow->second.cellList.begin(),itRow-
>second.cellList.end(),randomCell2), itRow->second.cellList.end());
			itRow->second.cellList.push_back(randomCell);
			//cout<<"first step completed"<<endl;

			itNode2->second = nodeSec2;
			itRow2->second.cellList.erase(std::remove(itRow2->second.cellList.begin(),itRow2-
>second.cellList.end(),randomCell), itRow2->second.cellList.end());
			itRow2->second.cellList.push_back(randomCell2);
		}
	}
```

```
	updateCenter();
	}

	void initiateMove2()
	{
	row r;
	map<string, node> ::iterator itNode, itNode2;
	map<int, row> ::iterator itRow, itRow2;
	bool accept;
	long int cost2,cost1,prevCost;
		cost1 = wireLength();
		cost2 = cellOverlap();
		prevCost = cost1 + cost2;

		int randRow = rand() % (rowId.size()) + 1;
		itRow = rowId.find(randRow);
		int rowCellNum = itRow->second.cellList.size();
		int choosenCell=rand()%rowCellNum, choosenCell2;
		itRow->second.cellList[choosenCell];
		if (choosenCell == 0)
			choosenCell2 = 1;
		else if (choosenCell == rowCellNum-1)
			choosenCell2 = rowCellNum - 1;
		else
			choosenCell2 = (rand()%2==0)?(choosenCell - 1):(choosenCell + 1);

		cout<<" exchange cell "<<itRow->second.cellList[choosenCell]<<" with cell "<<itRow-
>second.cellList[choosenCell2]<< endl;
		itNode = nodeId.find(itRow->second.cellList[choosenCell]);
		itNode2 = nodeId.find(itRow->second.cellList[choosenCell2]);
		if (itNode2->second.terminal==0)
		{
		r = itRow->second;

		int xTemp, yTemp;
		xTemp = itNode->second.xCoordinate;
		yTemp = itNode->second.yCoordinate;
		itNode->second.xCoordinate = itNode2->second.xCoordinate;
		itNode->second.yCoordinate = itNode2->second.yCoordinate;
		itNode2->second.xCoordinate = xTemp;
		itNode2->second.yCoordinate = yTemp;

		itRow->second.sortByX();

		updateCenter();
```

```
		accept = checkMove(prevCost);
		if(!accept)
		{
			itRow->second = r;
		}


		updateCenter();
		}
}

void update_Temperature()
{

if(Temperature>5000)
{
		Temperature=0.8*Temperature;
}
else if(Temperature<=5000 && Temperature >200)
{
		Temperature=0.94*Temperature;
}
else if(Temperature<200)
{
		Temperature=0.8*Temperature;
}
else if(Temperature<1.5)
{
		Temperature=0.1*Temperature;
}
float t = Temperature;
 cout<<" Temperature for this iteration : "<< Temperature <<endl;
}

bool checkMove(long int prevCost)
{
srand(time(NULL));
long int cost2=0,cost1=0,newCost=0;
//int cost3=0;
int delCost=0;
double factor=0;
double prob = rand()/(RAND_MAX+0.0);
			//cout<<"Checking validity of the move"<<endl;
			cost1 = wireLength();
```

```
                cost2 = cellOverlap();
                //cost3 = rowOverlap();
                newCost = cost1 + cost2;
                delCost = newCost - prevCost;
                factor = exp(-delCost/Temperature);

                if(delCost < 0)//|| prob < (exp(-delCost/Temperature)) )//这里做了修改，改为小
于
                {
                        prevCost = newCost;
                        //cout << "Cell position is updated " << endl;
                        return true;
                }
                else
                {
                        //cout << "Move is NOT valid, reverting" << endl;
                        return false;
                }
        }

        void timberWolf()
        {
                //int xLimit=macroPlacement();
                //cout<<xLimit<<endl;
                placeinitial();
                Temperature = 40000;
                int i;
                //int delCost;

                while (Temperature > 1 )
                {
                        i=3;
                        while(i > 0)
                        {
                                initiateMove(xLimit);
                                i--;
                        }
                        update_Temperature();

                }

        }

        void timberWolf2()
        {
```

```
	Temperature = 40000;
	int i;
	//int delCost;

	while (Temperature > 1 )
	{
		i=2;
		while(i > 0)
		{
			initiateMove2();
			i--;
		}
		update_Temperature();
	}
}

void showcell()
{
	map<string, node> ::iterator itNode;

	if(itNode != nodeId.end()){
	for(itNode = nodeId.begin(); itNode != nodeId.end(); ++itNode)
		{

			itNode->second.printParameter();

		}
	}
}

void showrow()
{
	map<int, row> ::iterator itRow;

	if(itRow != rowId.end()){
	for(itRow = rowId.begin(); itRow != rowId.end(); ++itRow)
		{

			itRow->second.printParameter();

		}
	}

}
```

Main.h

```
#ifndef MAIN_H
#define MAIN_H

#include<iostream>
#include<vector>
#include<fstream>
#include<iterator>
#include<algorithm>
#include<boost/algorithm/string/split.hpp>
#include<boost/algorithm/string.hpp>
#include<string>
#include <iostream>
#include <regex>
#include <map>
#include<xutility>

using namespace std;

class node;
map<string, node> nodeId;

int orientToInt(string a)
{
	int num=0;
	switch(a[0])
	{
	case 'N' : num = 0; break;
	case 'W' : num = 1; break;
	case 'S' : num = 2; break;
	case 'E' : num = 3; break;
	default : num = 4;
	}
	return num;
}

string intToOrient(int num)
{
	string a;
	switch(num)
	{
	case 0 : a = "N";break;
	case 1 : a = "W";break;
	case 2 : a = "S";break;
	case 3 : a = "E";break;
```

```
        default: a = "F";
        }
        return a;
}


 class node
{

public:

string nodeName;
int width;
int height;
int weight;
int terminal;//terminal 是什么
int xCoordinate;
int yCoordinate;
int xBy2;
int yBy2;
string orientation; //orientation 还没有想好怎么改
int cellRowId;
vector <int> Netlist;

void setParameterNodes(string nodeName, int width, int height,int terminal)
  {
        this->nodeName = nodeName;
        this->width = width;
        this->height = height;
        this->terminal = terminal;
  }

void setParameterWts(int weight)
  {
        this->weight = weight;
  }

void setParameterPl(int xCoordinate,int yCoordinate,string orientation)
{
        this->xCoordinate = xCoordinate;
        this->yCoordinate = yCoordinate;
        this->orientation = orientation;
}

void setOrientation(string orient)
```

```
{
        cout<<(this->orientation)<<"   "<<orient<<endl;
        this->orientation = orient;
        if ((orient!=this->orientation) && (this->orientation[0]!='F'))
        {
                int rotateDistance = (8 + orientToInt(orient) - orientToInt(this->orientation)) % 4;
                if ((rotateDistance%2==1)&&(this->width<16))
                {
                        int _width = this->width;
                        int _height = this->height;
                        this->width = _width;
                        this->height = _height;
                }
        }
}


void setRowId(int cellRowId)
{
        this->cellRowId = cellRowId;
}

void setNetList(int NetId)
{
        Netlist.push_back(NetId);
}

void setCenter(int xBy2, int yBy2)
{
        this->xBy2 = xBy2;
        this->yBy2 = yBy2;
}

void printParameter()
  {
        cout <<"nodeName      " << this->nodeName << endl;
        cout <<"Width         " << this->width << endl;
        cout <<"Height        " << this->height << endl;
        cout <<"Weight        " << this->weight << endl;
        cout <<"X_Co-ordinate " << this->xCoordinate << endl;
        cout <<"Y_Co-ordinate " << this->yCoordinate << endl;
        cout <<"X/2           " << xBy2 << endl;
        cout <<"Y/2           " << yBy2 << endl;
        cout <<"Orientation   " << this->orientation << endl;
        cout <<"cellRowId     " << this->cellRowId << endl;
```

```
        cout <<"terminal      " << this->terminal << endl;
        cout <<"NetList       " ;
        vector <int> :: iterator it2;
        for(it2=Netlist.begin(); it2 != Netlist.end(); ++it2)
        {
                cout << *it2 << " ";
        }
        cout << "\n" << endl;
  }
};


class row;
map<int, row> rowId;
int RowWidth;
int xLimit;
 class row
{

public:
int Id;
int coOrdinate;
int height;
int siteWidth;
int siteSpacing;
string siteOrient;
string siteSymmetry;
int siteRowOrigin;
int numSites;
int overlap;
vector <string> cellList;

void setId(int Id)

{
        this->Id=Id;
}


void setParameterRows(int coOrdinate,int height,int siteWidth,int siteSpacing,string siteOrient,string
siteSymmetry,int siteRowOrigin,int numSites)
  {
        this->coOrdinate = coOrdinate;
        this->height = height;
        this->siteWidth = siteWidth;
```

```
        this->siteSpacing = siteSpacing;
        this->siteOrient = siteOrient;
        this->siteSymmetry = siteSymmetry;
        this->siteRowOrigin = siteRowOrigin;
        this->numSites = numSites;
  }

void setCellList(string cellId)
 {
        cellList.push_back(cellId);
 }

vector<string> sortByX() //将cellList中的cell按x坐标排列
{
int i=0,x=0;
map<int,string> sortx;
map<int,string>::iterator it;
vector<string>::iterator itl;
vector<string>list;
for(i=0;i<(this->cellList.size());i++)
{
        x=nodeId.find(cellList[i])->second.xCoordinate;
        sortx.insert(pair<int,string> (x,cellList[i]));
}
for(it=sortx.begin();it!=sortx.end();++it)
{
        list.push_back(it->second);
}
this->cellList=list;
return this->cellList;
}

void calcRowOverlap() //
{
        vector <string> :: iterator it1;
        int xLast=0, widthLast = 0;

        xLast = nodeId[cellList[cellList.size()-1]].xCoordinate;
        widthLast = nodeId[cellList[cellList.size()-1]].width;
        overlap = xLast + widthLast - (RowWidth+xLimit);

        this ->overlap = overlap;
}
```

```
void printParameter()
  {
        cout <<"rowId         " << this->Id <<" "<< endl;
        cout <<"Row-Co-ordinate " << this->coOrdinate << endl;
        cout <<"height         " << this->height << endl;
        cout <<"siteWidth      " << this->siteWidth << endl;
        cout <<"siteSpacing    " << this->siteSpacing << endl;
        cout <<"siteOrientation " << this->siteOrient << endl;
        cout <<"siteSymmetry    " << this->siteSymmetry << endl;
        cout <<"siteRowOrigin   " << this->siteRowOrigin << endl;
        cout <<"numSites       " << this->numSites << endl;
        cout <<"Overlap of this row   " << overlap<<endl;
        cout <<"cellsInRow     " << " ";
        vector <string> :: iterator it1;
        for(it1 = cellList.begin(); it1 != cellList.end(); ++it1)
        {
                cout<< *it1 << " ";
        }
        cout << "\n" << endl;
  }
};

map<int, vector<string> > netToCell;

        #endif
```