EEDG 6375.001

DESIGN AUTOMATION OF VLSI SYSTEMS

Programming Assignment #2

SIMULATED ANNEALING

Abstract

Circuit partitioning is one of the important steps in VLSI physical design and involves the task of dividing a circuit into smaller parts for ease of design and layout. In the partitioning of circuit the main objective is to split the circuit into several sub-circuits such that the size of the subsystems are within prescribed ranges, the complexity of the interconnection and delay due to these interconnections between the subsystems are minimum. We know that circuit partitioning problem is NP-hard, that it does not have a polynomial-time algorithm to solve the problem and hence using various heuristics, approximate solutions are obtained in polynomial time.

In this project we implemented Simulated Annealing, a probabilistic method to bi-partition benchmark circuits. The algorithm-starts with generation of a random solution, a means of evaluating the problem functions, an annealing schedule- an initial temperature and rules for lowering it as the search progresses. We have tried to achieve the main objective of generating a bi-partitioned netlist with minimum cut size considering the area balance criteria by using appropriate data structures.

Introduction

Circuit partitioning is one of the important steps in VLSI physical design. The main objective is to split the circuit into several sub-circuits such that the size of the subsystems are within prescribed ranges, the complexity of the interconnection and delay due to these interconnections called the mincut problem, between the subsystems are minimum.

Simulated annealing belongs to the probabilistic and iterative class of algorithms. Hence, are capable of producing a different solution for the same problem each time they are used.

This algorithm is a simulation of the annealing process used for metals. The metal is heated to a very high temperature, so the atoms gain enough energy to break chemical bonds and are free to move. The slow cooling gives them more chances of finding configurations with lower internal energy than the initial one. If we compare the partitioning problem with the annealing process the attainment of a global optimum, e.g., the best cost is analogous to attaining a good crystal structure. In a two way partitioning problem, division of a circuit in a two equal sized blocks is a configuration with its cost. All other partitions obtained by swapping elements across the partitions represent a local neighborhood of a given state. If we want to reach the global optimum G, we must sometimes accept the worse solution to jump out from a local optimum L. Global optimum is obviously a state with the best cost. The simulated annealing algorithm simulates the annealing process at a given temperature. The procedure starts with an initial partition at an initial temperature Tstart. During the process, temperature is slowly reduced in a geometrical progression. At each temperature, a new partition is obtained at its cost, which is compared with the old cost. If the new cost is better, the new solution is accepted. If not, the inferior solution is accepted with a probability.

Algorithm

Simulated Annealing (SA) is a probabilistic, heuristic algorithm for the global optimization problem of locating a good approximation to the global optimum of a given function in a large search space. It can be viewed as a technique of iterative improvement in which an initial solution is improved repeatedly by alterations until a better solution is found. Simulated Annealing randomizes this procedure in a way that allows for moves (changes that worsen the solution) in an attempt to reduce the probability of becoming stuck at poor but locally optimum solution. SA can be readily adapted to new problems because of its apparent ability to avoid poor local optima, it offers hope of obtaining better results.

Step1: Initial bi-partition of circuit netlist is done considering area balance criteria

Step2: Cut set is calculated from the bi-partitioned netlists based on the number of interconnections between them

Step3: Swapping of certain number of nodes between the bi-partitioned netlist is performed and for each swap, new cut set is found

Step4: With the previous and new cut set value, standard deviation (sd) of the difference obtained by swapping is calculated

Step5: Initial temperature Tstart is calculated using the formula,

$$T_{start} = (-sd/\log(0.5))$$

Step6: ΛE= Function value of old partition – Function value of new partition. If

$\Lambda E > 0$, we accept the new solution and discard the previous one, since the function value i.e. cut size of new solution is less than the old one. If $\Lambda E <= 0$, we chose the new solution with probability p. The probability is

$p = e^{(\Lambda E/T)}$

where T=Tstart is temperature which is initially set by a large value and is decreased with time. As T decrease with time, the probability to accept the new solution with large cut size also decreases.

Step7: The number of swaps is decided based on the size of the input file and the initial temperature

Step8: The temperature decrement rule used is

$$T_{k+1} = \alpha T_k$$

Where, α =0.8 and the value is decremented with the cooling scheme

Step9: Run for SIZEFACTOR * N trials, observing the best solution seen and the average solution value. If either of these is better than the corresponding values for the previous SIZEFACTOR *N trials, repeat. Otherwise, lower the temperature in the standard way

Step10: The final best cut set is obtained by running a number of iterations where the number of annealing runs is performed with the parameters, Tstart (initial temperature), size of input file, cooling ratio α.

Implementation

We have implemented Simulated Annealing in C++ programming language. The code is run and tested on the UTD engnx Linux server using NX Client. A detailed description of the data structures and the code used in implementing the algorithm is given below.

The code is named *Naidu_GhatySreesh_pa2.cpp*. The netlist and the cell list is stored in Hash tables. The class *HashTable* is used to declare and define the hash table. This class in turn includes pointer to the class *LinkedList* which holds the values of nets and nodes in netlist and cell list respectively.

The function *insertNode()* inserts a new node to the hash table. The function *printTable()* prints the netlist. The *printcelllist()* function prints the cell list.

The function *to_key()* gives a unique hash key based on the node name. It is programmed in such a way that each node has a unique hash key in order to avoid collision.

In the *main()* function, the .net and .are file names are accepted from the command prompt. Using the .net input file and the functions of HashTable and LinkedList classes called through its corresponding objects, the netlist and cell list are saved based on the occurrence of the letter ‘s’ in the netlist. The areas of each node are saved to the cell list by reading it from the .are file.

Based on the areas of the nodes, they are pushed into 2 vectors called part1 and part2. This push happens such that the node is added to the part with lesser area.

The function *cutset()* accepts the two partitioned vectors, the netlist and the cell list as parameters to calculate the cut set between the 2 partitions (Cut set is the number of connections between the two partitions). Using this function, the initial cut set is calculated.

In order to find the initial temperature Tstart, we swap first 50 nodes between the 2 partitions or the size of the smaller partition, whichever is small. After each swap, the cut set is found and the standard deviation of the difference (between each swaps) is calculated. The standard deviation is stored in variable ‘sd’. The Tstart is calculated by (-sd/log(0.5)).

The swaps are made on the initial partition / previously accepted partition and the new partition is accepted if the difference (cut old-cut new) is positive or if negative, it is accepted with probability given by a random value generated by rand() function.

The number of swaps to be made at each temperature is varied based on the size of the initial partition and also the current temperature. The number of swaps are reduced as the size of the file increases, also as the temperature decreases. The swap is made only if the areas of the new partitions have a ratio between 0.3 and 0.7.

The variation of the temperature depends on the current temperature and its value compared to Tstart. As the temperature decreases, the multiplier value is decreased. In other words, as temperature is low, it is decreased at even faster pace.

The calculation is terminated once the operating temperature goes below a value given by (Tstart/10)

At each temperature, the accepted cut set values along with the area ratio are printed to the output file.

The final minimum cut set value and the final partitions are printed to the output file named Naidu_GhatySreesh.txt.

This code is included between the lines provided in order to print the execution time.

<u>Results:</u> The following table gives the results obtained from bi-partitioning

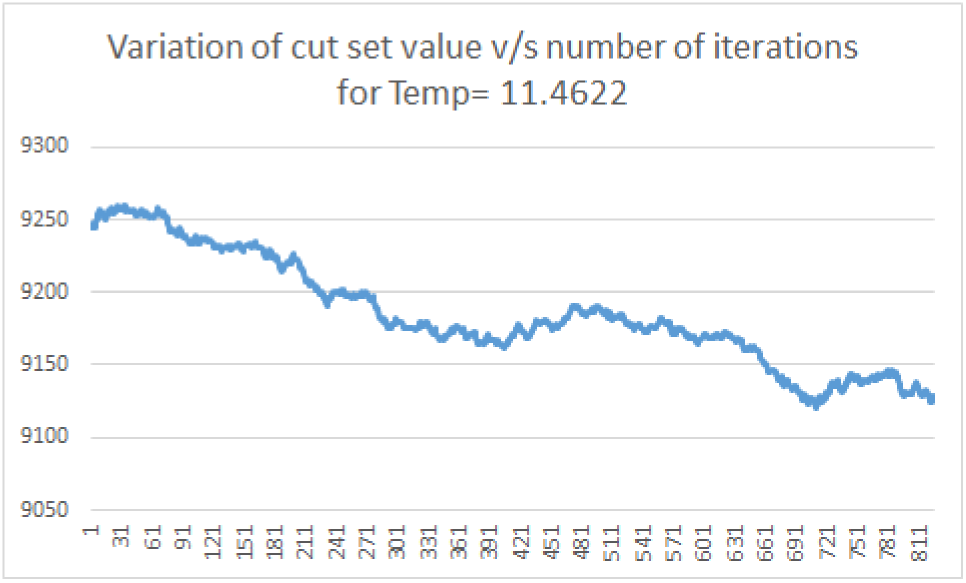
Benchmark name	Execution time(seconds)	Initial cut	Final cut	Percentage change	Ratio cut
EACG21	0.2029	9	7	22.3	29.866
EACG50	1.2678	25	21	16	97.32
IBM01	36908.587	9249	7994	13.569	32816.09
IBM03	50220.751	17396	15408	11.427	63791.73
IBM04	61956.275	20564	16428	20.113	65817.30
IBM06	67248.622	23111	19085	17.420	86318.40
IBM08	76862.816	32875	29615	9.917	119043.3

IBM10	96912.874	49570	44443	10.35	182323.5
IBM12	117286.39	51891	47704	8.068	210814.7
IBM14	128627.98	101768	92764	8.847	398002.3
IBM16	135082.56	129534	120563	6.947	501518.3
IBM18*	176400	139348	132118	5.188	607159.9

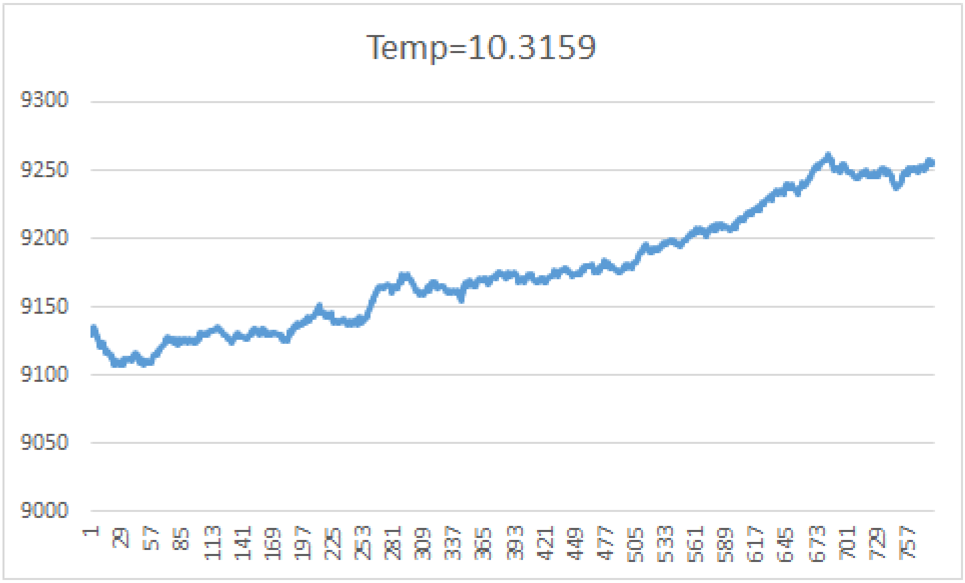
*Process still running. Values in the table are values obtained in the file just before submission. (Time is an approximate value since the time we started the run until submission)

The results obtained after running IBM01 are tabulated and the following graphs are obtained for different temperatures. As the temperature is decreased, the number of iterations are also decreased. The x-axis depicts the number of iterations, and y-axis depicts the cut set value. It can be seen that at higher temperature, good solutions as well as bad solutions are accepted. As temperature is decreased, only good solutions are accepted.

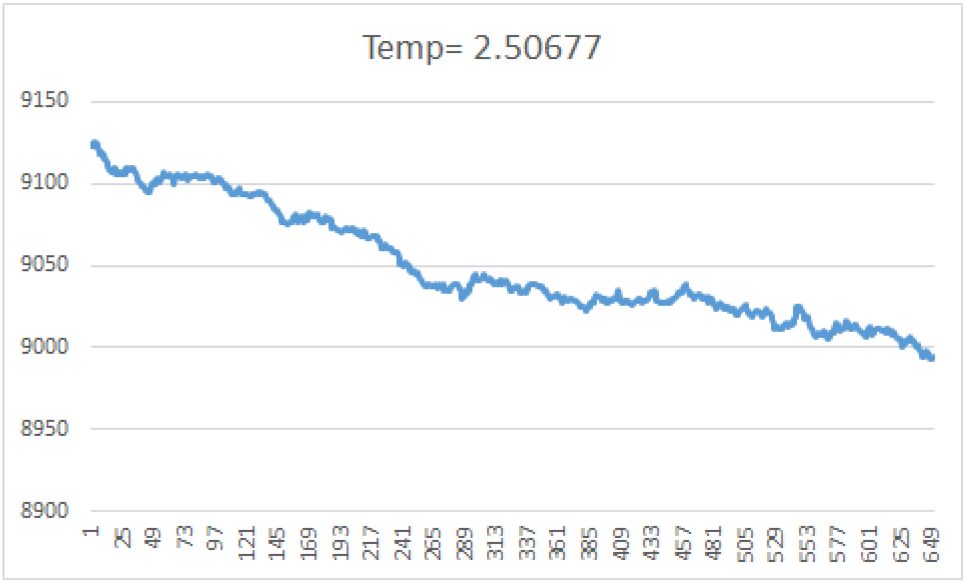
a) Temp=11.4622



b) Temp=10.3159



c) Temp=2.50677



Conclusion

- Observation: SA algorithm is showing better results only in certain cases or circuits. Otherwise, cut-set becomes larger than the previous cut-set taken. This is because the parameters taken are constant for all the benchmarks.
- Disadvantages: Repeatedly annealing with a 1/logn schedule is very slow, especially if the cost function is expensive to compute.

  Also, this method cannot tell if an optimum solution is found. Some other method (such as branch and bound) should be used to accomplish this.
- Scope for improvement: The results may be different and better if different parameters are chosen for different circuit configurations. For example, if the tolerance level and no of nodes to swap parameters are increased for the large nodes circuit, it may give better results.
- Simulated Annealing is not the best to circuit partitioning. However, SA algorithms are better than greedy algorithms, when it comes to problems that have numerous locally optimum solutions. Simulated Annealing guarantees a convergence upon sufficiently large number of iterations.

References

[1] Siddhartha Banerjee and Bibek Ranjan Ghosh, '*Circuit Bipartition Using Simulated Annealing*', IJAEGT.

[2] Aalay Kapadia and Dr. Dinesh Bhatia, '*Survey of Special topics for Circuit Partitioning'.*

[3] Arijit Bhattacharya, Sayantan Ghatak, Satrajit Ghosh, Rajib Das, '*Simulated Annealing approach onto VLSI Circuit Partitioning'.*

[4] *What is a HashTable Data Structure* - *Introduction to Hash Tables,* https://www.youtube.com/watch?v=MfhjkfocRR0

[5] Zoltan Barucha, Octavian Cret, Kalman Pusztai, *'Comparitive study of circuit partitioning algorithm'.*