AN EMPIRICAL STUDY OF THE STOCHASTIC EVOLUTION ALGORITHM FOR THE VLSI CELL PLACEMENT PROBLEM

by

Natrajan Thamizhmani

A project submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

February 2008

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I sincerely thank Dr. Year Back Yoo, my advisor, for providing me with guidance to my research and helping me with it throughout my research process. I also extend my gratitude to the faculty and staff members of the Department of Computer Science who have helped me throughout my graduate education.

I would also like to thank my parents, Thamizhmani Thambi and Mangalam Ganapathy, for their never ending love and affection and providing me with everything I have now and encouraging me to pursue my graduate education; my grandmother, Lakshmi Ganapathy for her love and affection and for being my inspiration. Thanks to my sister, Lakshmi Thamizhmani for her love and affection, support, friendship, encouragement and for being my role model since my childhood; and God, for making things possible.

ABSTRACT

The Stochastic Evolution (SE) algorithm is a relatively new heuristic method that is used for combinatorial optimization that exploits an analogy between biological evolution and combinatorial optimization.

The SE algorithm begins with a random initial solution or with a previously found good solution to the problem and simulates the evolution process by eliminating the bad characteristics of the older generation resulting in an improved newer generation solution. The SE algorithm achieves this using functions and operations which test the suitability of characteristics for the existing environment. Each characteristic of a species in the current generation has to prove its suitability under the existing environmental conditions in order to remain unchanged in the next generation. This process is repeated until a certain number of iterations is completed or until no significant improvement is noticed and the solution to the problem is obtained.

In this project, the SE algorithm is studied and implemented to solve the very large scale integration cell placement problem, and the quality of the solutions and the running times of the algorithm are compared with those generated by the Simulated Annealing (SA) algorithm.

The SE algorithm after experiments shows that it produces results that are comparable to the results that were generated by the SA algorithm. The SE algorithm seems to be suitable in cases where the size of the input is considerably large. The SE algorithm starts consuming more time than the SA algorithm as the size of the input increases. The feature in the SE algorithm which increases the number of trials if the newer generation is better than the older could increase the running time of the SE algorithm considerably.

CHAPTER 1

INTRODUCTION

Concept of Evolution

According to Darwin's theory of evolution, evolution is a slow gradual process that acts by taking advantage of slight successive variations. Certain characteristics that are inherited from one generation to the next are slightly changed over time. Such changes among population could be because of various reasons. Individuals of a population undergo such changes to adapt themselves and become well suited to their existing environment. Natural selection is the process by which characteristics that are useful for the population in the existing environment are retained and less useful characteristics start to disappear from the individuals in the population. Individuals in a population who can better adapt to the conditions and survive reproduce and breed more successfully [1]. With time the individuals adapt and undergo changes in traits which happen as a result of the change in their genes from one generation to the other. The individuals who have the ability to adapt and survive end up living for successfully longer time periods.

Stochastic Evolution Algorithm Overview

In 1990, Youssef G. Saab and Vasant B. Rao at University of Illinois proposed the stochastic evolution algorithm in their publication *Stochastic Evolution: A Fast Effective Heuristic for Some Generic Layout Problems.* The Stochastic Evolution (SE) algorithm is an efficient and easy-to-implement heuristic which has been applied successfully to the traveling salesman problem and the network bisection problem.

The SE algorithm takes a random initial configuration, or a previously known good solution as its input. The characteristics of a species in the current generation have to prove their suitability under the existing conditions in order to be retained for the next generation. The cost involved is calculated during each evolution. If an improvement is found, the remaining number of trials is increased as an incentive. A cost increasing move is occasionally accepted stochastically. Thus, during each evolution, we accept all cost improving evolutions, and stochastically accepting steps that do not improve the cost ensures that the algorithm does not get caught in a local minimal value. The algorithm aims for a better solution during each iteration and continues until a specific number of trials is reached or there is no improvement over a certain period of time. The solution obtained when the algorithm terminates is taken as the solution to the problem.

Experimental Goals

This project aims to implement the SE algorithm on a combinatorial optimization problem to examine the efficiency of this relatively new heuristic method. We choose the VLSI cell placement problem and we compare the efficiency of the SE algorithm with a well known heuristic, Simulated Annealing (SA) [2]. Not much experimental study has been done in comparing different heuristics for the VLSI cell placement problem.

We implement both algorithms on the same computer system using the same programming language. We then compare the quality of the solutions as well as the execution time on the benchmark dataset inputs.

Hardware and Software

All the programs were run on a standalone machine which had the Intel Pentium Dual CPU T2310 processor and had 1 GB of RAM and ran the 32 bit version of the Windows Vista Operating System.

Java was used as the programming language to implement the SE and the SA algorithms using Eclipse – an open source development platform.

CHAPTER 2

HEURISTICS FOR COMBINATORIAL OPTIMIZATION PROBLEMS

Combinatorial Optimization Problems

Combinatorial optimization problems are problems which have a discrete set of possible solutions. These problems seek a global minimum among these various possible solutions. Some well known combinatorial optimization problems include the network bisection problem, the traveling salesman problem, the job scheduling problem and the VLSI cell placement problem. Such problems have a global minimum solution which most heuristics seek to reach. However, all these problems are known to be NP-complete. This would mean that an enormous amount of time would be required to solve or compute and find the optimal solution through an exhaustive search.

Why use heuristics?

Heuristics are simple sets of rules that aim at solving a hard problem. Several heuristic methods are used in day to day life with or without realizing them. Heuristics provide a description of the successive stages of a decision process. The heuristic might often involve a decision making step at a certain stage during the search for a solution [3, 4]. Well chosen heuristics work pretty efficiently and can give solutions close enough to the global optimal solution in a reasonable amount of time. Therefore they have proved to be time saving methods that give acceptable results as compared to exhaustive searches which could require exponential time to obtain the optimal solution of the problems.

Modern Heuristics

There are a variety of modern heuristics that have been applied to combinatorial optimization problems. Some of these heuristics include the widely used and well known SA method, the Tabu search, various genetic algorithms, ant colony optimization, stochastic evolution and simulated evolution [2, 7].

The SE algorithm is one of the several heuristics which helps in solving hard problems relatively quickly as compared to exhaustive search. The SE algorithm is described in detail in Chapter 3. However it is not a rule that a near-optimal solution is always guaranteed while using heuristics.

Simulated Annealing Algorithm

Simulated annealing algorithm is a general adaptive heuristic and the best known method for module placement. Though it is known to be a time consuming method, it yields excellent results. It works with most of the combinatorial optimization problems. The SA algorithm is also a non-deterministic algorithm and is robust in nature. One typical feature of the SA algorithm is that, besides accepting solutions with improved cost, it also, to a limited extent, accepts solutions with deteriorated cost. This algorithm also involves some parameters that play a vital part in the execution of the algorithm. It is also easy to implement [2, 7].

The basic procedure in the SA algorithm is to accept all moves that result in a cost reduction. Certain moves that result in a cost increase are accepted probabilistically. A parameter T, called the *temperature* is used to control the acceptance probability. $\Delta Cost$ is the difference in costs before and after the move was made.

The acceptance probability, A_{ij} can be summed up as follows [7].

$$A_{ij} = \begin{cases} e^{-\Delta Cost/T} & \text{if } \Delta Cost_{ij} > 0 \\ 1 & \text{if } \Delta Cost_{ij} \leq 0 \end{cases}$$

Here, *i* and *j* represent the previous and current state respectively

Advantages and Disadvantages

The SA is a robust and easy to implement technique. This algorithm can be used to implement various combinatorial optimization problems. It provides solutions of reasonably good quality. It is popular and the most widely used algorithm.

The algorithm involves various parameters which need to be set appropriately to get reasonably good solutions. It is also a well known fact that a great deal of computation time will be needed for finding solutions using the SA algorithm.

CHAPTER 3

STOCHASTIC EVOLUTION ALGORITHM

Algorithm definition

Stochastic evolution is a powerful general and randomized iterative heuristic for solving combinatorial optimization problems. The algorithm was proposed by Youssef Saab and Vasanth Rao in 1989. It is stochastic because the decision to accept a move is a probabilistic decision. Moves that improve the cost function are accepted with probability one, and bad moves may also get accepted with a non-zero probability. The SE algorithm is a non-deterministic algorithm, an algorithm with one or more choice points where multiple continuations are possible and the choice point taken is not known ahead of time. The word evolution is used in reference to the evolution processes of biological species [7].

Combinatorial optimization problems can be modeled in a number of ways. SE adopts the following generic model:

> *Given a finite set M of movable elements and a finite set L of locations, a state is defined as a function S: M → L satisfying certain constraints* [7].

The SE algorithm includes the steps shown in Figure 1. Though it shows the outline of the algorithm, there can be slight modifications based on the type of problem it implements. We shall examine and describe in depth the steps of this algorithm in the following section.

Algorithm SE (S_0, p_0, R)

```
Begin
        BestS = S = S0;
        BestCost = CurCost = Cost(S);
        p = p0;
        ρ = 0;
        Repeat
                PrevCost = CurCost;
                S = PERTURB(S,p);
                CurCost = Cost(S);
                UPDATE (p,PrevCost,CurCost);
                If (CurCost < BestCost) Then
                        BestS = S;
                        BestCost = CurCost;
                        ρ = ρ – R;
                Else
                        ρ = ρ +1;
                EndIf
        Until ρ > R
Return (BestS);
End
```

Figure 1: General outline of the SE algorithm [7]

As we can see from the algorithm the inputs to the SE algorithm are:

1. an initial state (solution) S_0,
2. an initial value p_0 of the control parameter p, and
3. a stopping criterion parameter R.

Throughout the search for the optimal solution, S holds the current state (solution), while *BestS* holds the best state. If the algorithm generates a worse state, a uniformly distributed random number in the range $[-p, 0]$ is drawn. The new uphill state is accepted if the magnitude of the loss is greater than the random number, otherwise the current state is maintained. Therefore, p is a function of the average magnitude of the uphill moves that the algorithm will tolerate. The parameter R represents the expected number of iterations the algorithm needs until an improvement in the cost with respect to the best solution seen so far takes place. Finally the variable ρ is a counter used to decide

when to stop the search. ρ is initialized to zero and R - ρ is equal to the number of remaining generations before the algorithm stops [7].

Figure 2 shows the general outline of the PERTURB and the UPDATE methods which are two very important procedures of the SE algorithm.

```
FUNCTION PERTURB(S,p)

Begin
        ForEach (m ϵ M) Do
                S′ = MOVE(S,m);
                Gain (m) = Cost(S) – Cost(S′);
                If (Gain (m) > RANDINT (-p, 0)) Then
                        S = S′
                EndIf
        EndFor;
        S = MAKE_STATE(S);
        Return(S)
End

PROCEDURE UPDATE (p, PrevCost, CurCost);

Begin
        If (PrevCost = CurCost) Then
                p = p + pincr;
        Else
                p = p0;
        EndIf;
End
```

Figure 2: Perturb and Update methods [7]

Initial Placement

It is well known that each state is a potential solution in a combinatorial optimization problem. The initial placement is also a potential solution to the placement problem starting from which we seek better results. The starting placement is either a previously well known good placement or a random placement. We, for comparison reasons, create an initial placement which is common for both the SE and the SA algorithms.

Selection

During the selection phase of the algorithm we determine whether or not a cell should remain in its current location. Each of the movable elements is moved around and the current cost is compared to the previous cost before the move was made and thus by comparing the costs we can decide whether the cell deserves to stay in its current location or if moving the cell to another location will yield a better cost.

The *Gain* value, which was calculated to be the difference in the cost before and after the move was made, is compared to the random integer that is generated between *0* and *-p* and if the *Gain* value is more than the generated integer, the new state is retained.

$$Gain\ (m) = Cost(S) - Cost(S')$$

This step will only ensure that all moves that prove to cause a decrease in the cost after a move is made will always be retained and moves that cause a negative *Gain* value

will probabilistically be accepted [7].

Control Parameter (*p*)

The control parameter, *p*, plays a significant role in the algorithm. This parameter ensures that the algorithm doesn't get caught in a local minimum cost value. Even when it does get caught, we accept negative *Gain* values probabilistically. This is done by comparing the *Gain* value with the random number generated. The control parameter plays an important role in the generation of this random number. Each time after the *PERTURB* function is called the control parameter value is updated. If the cost remained same after the function call, the *p* value is increased by *pincr,* a predefined value; otherwise, it is set to the initial *p* value. This will ensure that the range in which the random integer is generated is expanded and it will be more likely that the algorithm will escape the local minimal solution in the following iterations.

Termination condition

The SE algorithm will have to stop at some point and provide a solution to the problem. This can be decided by doing one of the following.

1. Allowing the algorithm to run until a specific number of iterations, *R*, are completed
2. Terminate the algorithm when there is no cost improvement over a specific number of generations.

The more commonly used method is to run the algorithms over a predefined number of iterations *R*. The selection of this parameter determines the running time of the algorithm and therefore should be done carefully. A very high *R* value will result in the algorithm running for longer than needed. A low value will result in not giving enough time to the algorithm to improve the initial state. Therefore it is important to choose an optimal value for this parameter. Depending on the problem, the type of termination is decided.

Theoretical Foundations

SE has been very successful at solving problems like the traveling salesman problem and the network bisection problem. Youssef G. Saab and Vasant B. Rao, who proposed the SE algorithm, have suggested that the SE algorithm works better than the SA algorithm. Over time, the solution improves because the cells which are already well placed retain their locations during subsequent iterations and other cells try to move to better locations and consequently improve and produce a better solution.

Advantages and Disadvantages

The SE algorithm is used to solve a wide variety of combinatorial optimization problems. This algorithm is adaptive; it uses a set of control variables that can be modified to adapt better to the particular problem being solved. It has also proved to yield better quality solutions at a faster execution speed in comparison to other algorithms and evolution-based methods. SE is well suited for problems of large size [6].

Modeling the states of the problem is a major challenge. Setting the initial values for the parameters used in the algorithm is difficult. The value of the control parameter p should not be too low or too high. A low value will result in certain moves not being performed and a high value will result in large negative *Gains* being accepted. Therefore an appropriate value selection for p is crucial. While selecting the total number of iterations, R, we should again be sure it is not too low or too high. A low value might cause the search for the solution not to be done properly and a high value might result in unnecessary execution of the algorithm in the later stages [6].

CHAPTER 4

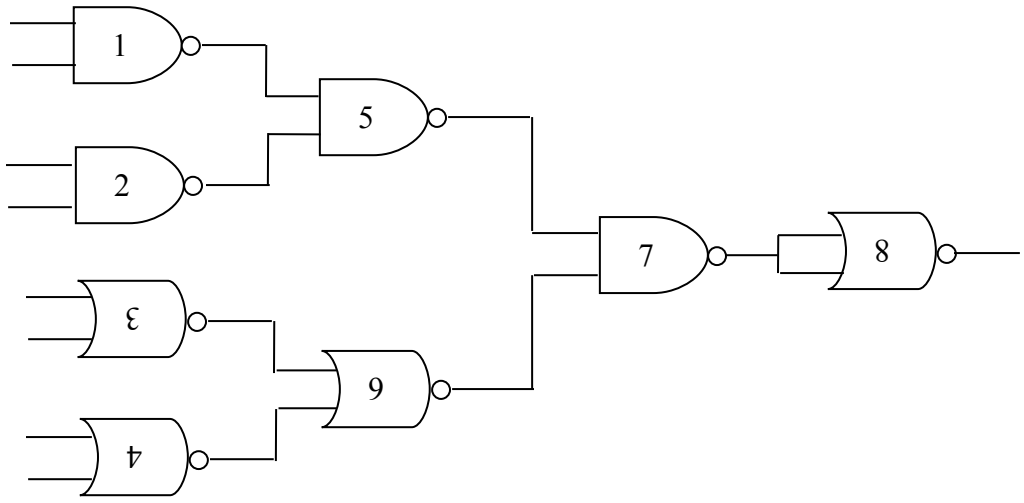
VLSI CELL PLACEMENT PROBLEM

Problem Description

Placement is the process of arranging circuit components on a layout surface. The placement problem is a generalization of the quadratic assignment problem, which is NP-complete. The placement problem can be defined as follows [7].

> *Given an electrical circuit of modules with predefined input and output terminals and interconnected in a predefined way, construct a layout indicating the positions of the modules so the estimated wire length and layout area are minimized.*

The inputs to the problem are the module description, consisting of the shapes, sizes, and terminal locations, and the *netlist*, describing the interconnections between the terminals of the modules. The output is a list of *x*- and *y*-coordinates for all modules.



(a)

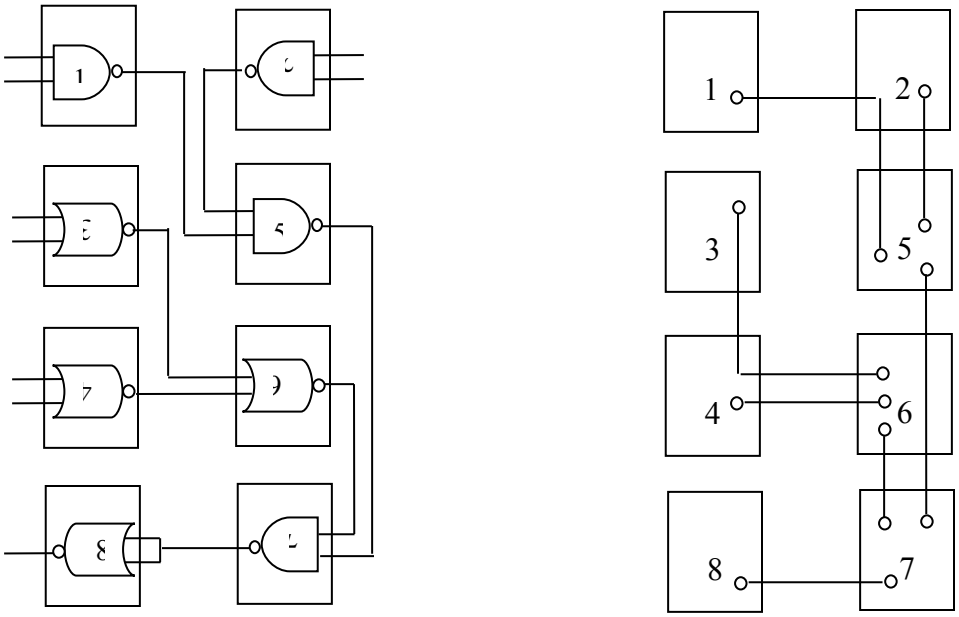


Figure 3: (a) A tree circuit. (b) A 2-D placement of gates. (c) A 2-D symbolic placement

Consider the circuit of Figure 3(a); suppose that we need to place the gates on a two-dimensional surface. One such placement is shown in Figure 3(b), Figure 3(c) represents an equivalent symbolic representation. In the symbolic representation, we can observe that the gates are represented as boxes and the nets as black lines [7]. It is possible to estimate the wire length from the symbolic representation. The area of a layout comprises the functional area and the wiring area. The functional area is the total of all the areas of the functional cells. The functional area remains unchanged for all placements. It is the wiring area which changes with the placement. This is because of the minimum separation that must be maintained between two wires and between a wire and a functional cell [7].

A placement that requires a large amount of wiring space must necessarily involve long wires and hence a large value of total wire length. The overall wire length for a placement *P* can be calculated using methods like the semi perimeter method etc.

A placement consists of *nodes*, *terminals* and *nets*. *Nodes* are the cells that can be moved and placed in the available locations in the layout board. *Terminals* are cell that are immovable. They have a predefined location in which they are seated. *Nets* are simple rules that define which cells should be interconnected.

Checkerboard model

A placement problem in which the cells are assumed to be squares and of equal size and all terminals are assumed to be at the center of the cell is called the checkerboard model. The length of the interconnection from one cell to the next is one unit. There are also no horizontal row spaces.

Placement Algorithms

The main objectives of a placement algorithm are to minimize the total chip area and the total estimated wire length for all the nets. The chip area usage has to be optimized in order to fit more functionality into a given chip area. The wire length has to be minimized in order to reduce the capacitive delays associated with longer nets and speed up the operation of the chip [2].

Placement algorithms can be classified into two major classes: constructive placement and iterative improvement. In constructive placement, a method is used to build up a placement from scratch; in iterative improvement, algorithms start with an initial placement and repeatedly modify it in search of a cost reduction.

Other possible classifications for placement algorithms are deterministic algorithms and probabilistic algorithms. Algorithms that function on the basis of fixed connectivity rules or formulas or determine the placement by solving simultaneous equations are deterministic and will always produce the same result for a particular placement problem. Probabilistic algorithms, on the other hand, work by randomly examining configurations and may produce a different result each time they are run. Constructive algorithms are usually deterministic, whereas iterative improvement algorithms are usually probabilistic [2].

Initial placement configuration

The initial configuration that could be used for the placement problem can be a previously known good placement or it can be a random placement. Using previously known good configurations often have proved to produce better solutions. Placements can be generated by randomly assigning the movable cells to the available locations on the layout board.

Selection

During the selection step, the set of movable cells is scanned and ordered in a

specific fashion. The ordering could be based on the number of nets the cells are involved in or it could be a random ordering. In some algorithms, like the SE algorithm [8], during the selection step the cells are arranged in decreasing order of the degree, where the degree represents the number of nets in the net list the cell is connected to.

Move operation

The move operation can be of two types: *Simple* move or *Compound* move. Given a function $S: M \rightarrow L$ and a movable element $m \in M$, a simple move from S with respect to m is just a change in the value of $S(m)$, i.e., a *simple* move generates a new function $S': M \rightarrow L$ such that $S'(m) \neq S(m)$ while $S'(m') = S(m')$ for all $m' \neq m \in M$. A *compound* move is a sequence of *simple* moves. Move operations are also sometimes defined based on the problem and are often altered based on requirements [6, 16].

Semi-perimeter method

The semi-perimeter method is one of the well known methods for wire length estimation. In this method the perimeter of the smallest rectangle that connects the pins of the cells of a net is calculated. Half of the perimeter calculated gives the length or cost of the wire needed for that net to be used in the chip. Since it is efficient it is a commonly used method for wire length estimation. For nets with more pins, the estimated wire length using this method will be less than the actual wire length that will be needed. However, in comparison to other wire length estimation methods like Steiner tree, minimal spanning tree, chain connections, and others this method gives a fairly good estimate. An example of how the semi-perimeter is calculated [2] is shown in Figure 4.

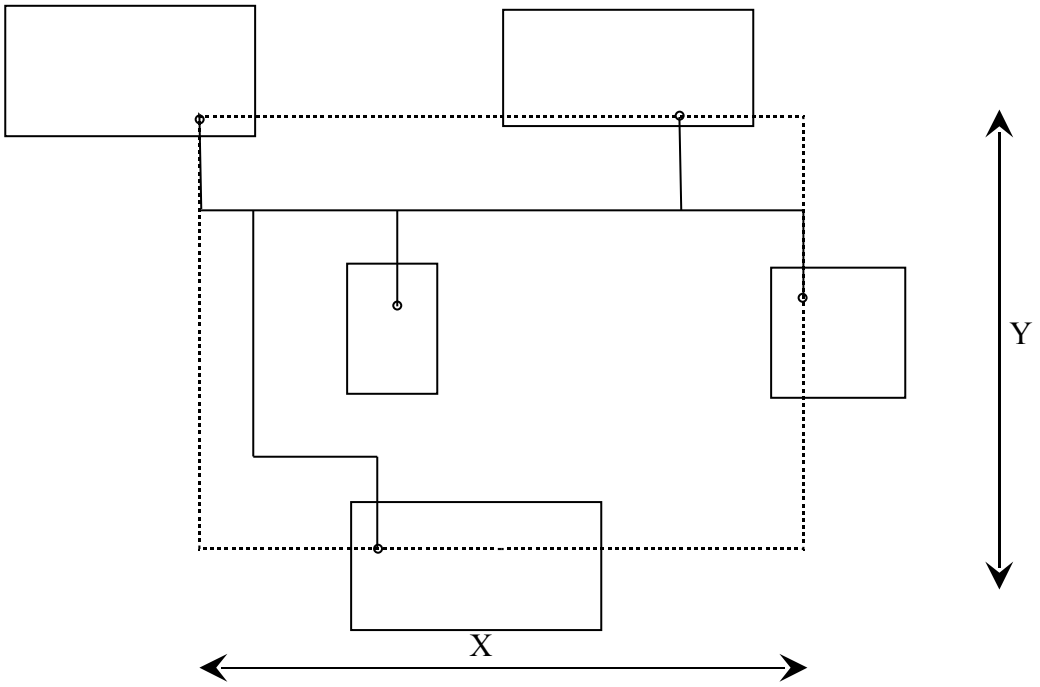


Figure 4: Semi-perimeter wire length = X + Y.

CHAPTER 5

EXPERIMENTAL RESULTS

Data Sets

To test the effectiveness of the SE and SA algorithms on the VLSI cell placement problem, datasets were selected from various sources. The files selected were *Cross, MPIO, SubOpt_1, SubOpt_4* and *p1UnitWDims.* Each of these files contains the details about the placement problem. The number of movable nodes, the number of nets and the net details are found in these files. These were selected as the inputs for the placement problem because these inputs were of various sizes. All inputs were of bookshelf format.

The datasets that were used were of diverse types and have nodes and nets ranging between a few movable cells to a few thousand movable cells. The number of nodes and nets in the input files are given in the Table 1. Each of these variations was tested a total of 10 times. Each of these placements represents different types of placement.

Input file name	*SubOpt_4* *(A)*	*Cross* *(B)*	*SubOpt_1* *(C)*	*p1UnitWDims* *(D)*	*MPIO* *(E)*
Nodes	280	386	500	833	2404
Terminals	30	26	252	81	4
Nets	485	681	252	902	4800

Table 1: Details of the input files [14, 15]

The input files *SubOpt_4, Cross, SubOpt_1, p1UnitWDims, MPIO* are referred to as A, B, C, D and E respectively.

Experimental Results

The SE and the SA algorithms were run against the VLSI cell placement problem for the various datasets mentioned in Table 1. As already mentioned in the previous section, the datasets used were of various sizes and types. Each of these inputs was implemented for both the algorithms and tested for a total of 10 times, and their average values were recorded.

The average results obtained for both the algorithms after the experiments were conducted for 10 trials are shown in Table 2.

Input File	*Wire length*		*CPU Time*	
Name	*SE*	*SA*	*SE*	*SA*
A	1445	1303	137	164
B	3215	2891	206	227
C	9764	9778	193	177
D	6790	7225	258	231
E	6923007	6923642	474	384

Table 2: Experimental results comparing SE and SA

The wire lengths for the SE and the SA algorithm were obtained for these datasets and compared in terms of the quality of their solution and the time taken by the algorithms to compute the wire length and the placement. The CPU time is represented in minutes. The algorithms were run for almost the same time and the results were obtained. The CPU times of the SE and the SA algorithm are fairly comparable. It can be noted that the CPU time of SE algorithm increases with increase in the size of the input.

The results obtained from the not-so-very-popular SE algorithm are comparable to the results obtained from the widely known and used SA algorithm. As the graph in Figure 5 indicates, the SE algorithm produces results comparable to the results produced by SA algorithm. The heights of the blocks in the column graph of Figure 5, lie closely to each other thus indicating how close and comparable the two methods are.

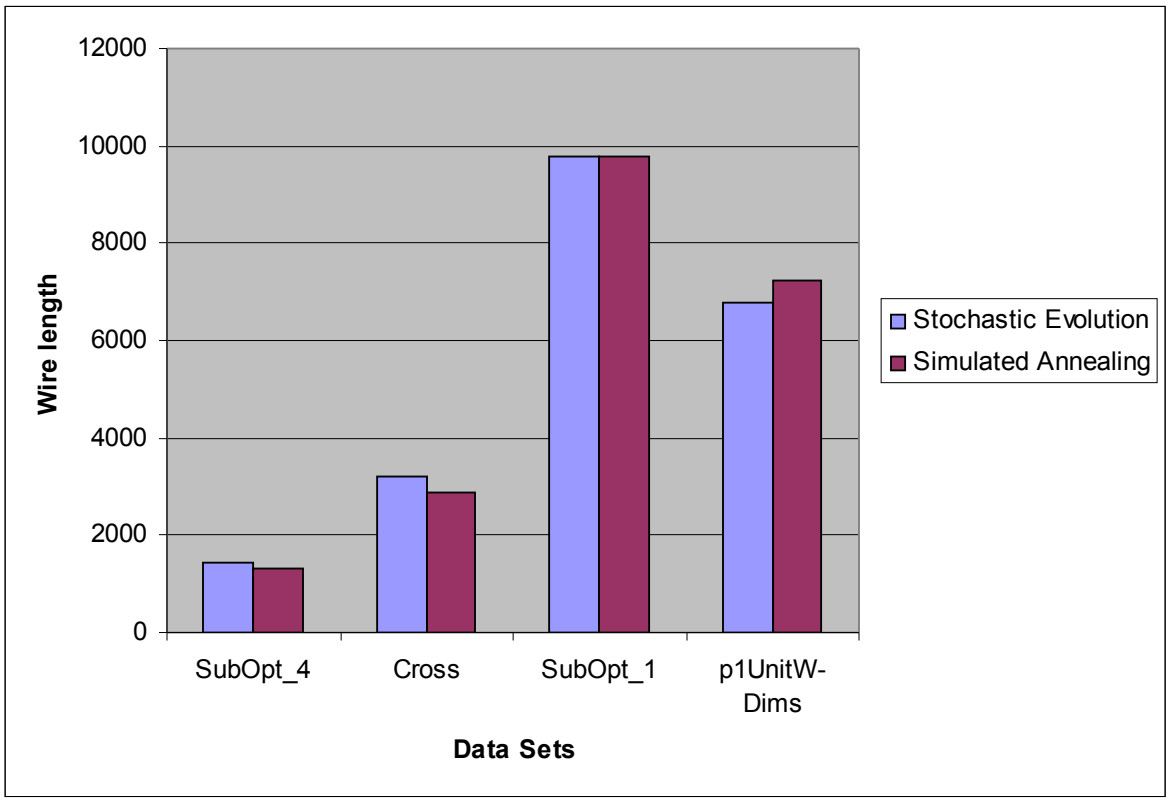


Figure 5: Column graph comparing SA and SE

CHAPTER 6

CONCLUSION AND FUTURE DIRECTIONS

Concluding Remarks

From the experimental results shown in Chapter 5, we can observe that the SE algorithm competes well with the SA algorithm. The solutions produced by the less known SE algorithm were comparable to the most widely used popular SA algorithm. Though it cannot be concluded from the results that the SE algorithm is the superior algorithm, given almost same amount of time the two algorithms produce almost same quality solutions. It can also be noted that with increasing input size, the SE algorithm takes more CPU time and produces slightly better quality solutions. This suggests that the SE algorithm might be well suited for large sized inputs and problems. This also suggests that the SE algorithm can be applied for various other combinatorial optimization problems too and can produce solutions of good quality.

Future Directions

The results indicate that the SE algorithm could be applied for various other combinatorial optimization problems and the heuristic could give results similar to the simulated annealing algorithm.

Parallelizing the SE algorithm can produce better results than the sequential version of the SE algorithm. This is because when the solution space is searched in parallel and the current best solution amongst the results obtained is used as the starting point for

further steps, this would result in a faster search and could produce better quality solutions.

References

[1] On the Origin of Species by Charles Darwin
A Facsimile of the First Edition with an Introduction by Ernst Mayr

[2] VLSI Cell Placement Techniques - K. SHAHOOKAR AND P. MAZUMDER
ACM Computing Surveys, Vol 23, No 2, June 1991

[3] Heuristics and Biases – The psychology of Intuitive Judgment
Edited by Thomas Gilovich, Dale Griffin, Daniel Kahneham

[4] Simple Heuristics that make us smart Gerd Gigerenzer, Peter M Todd and the ABC Research Group

[5] Y.Saab and V.Rao. An evolution-based approach to partitioning ASIC systems. 26th ACM/IEEE Design Automation Conference, 1989. Pg 767-770

[6] Y.Saab and V.Rao. Stochastic evolution: A fast effective heuristic for some generic layout problems. 27th ACM/IEEE Design Automation Conference, 1990. Pg 26-31

[7] Iterative Computer Algorithms with Applications in Engineering – Solving Combinatorial Optimization Problems by Sait and Youssef

[8] R. Kling and P. Banerjee, "ESP: Placement by simulated evolution," IEEE Trans. Computer- Aided Design, vol. 8, no. 3, pp. 245-256, Mar. 1989

[9] J. Cohoon and W. Paris, "Genetic Placement," Proc. IEEE International Conference On Computer-Aided Design, pp. 422-425, 1986.

[10] S. Nahar, S. Sahni, and E. Shragowitz, "Simulated Annealing and Combinatorial Optimization," Proc. 23rd Design Automation Conference, pp. 293-299.1986.

[11] B. Dunham, D. Fridshal, R. Fridshal, and J.North, "Design by Natural Selection," Synthese, D. Reidel Publication Company, Dordrecht-Holland, pp. 254-259. 1963.

[12] A. Dunlop and B. Kemighan, "A Procedure for Placement of Standard-Cell VLSI Circuits," IEEE Trans. Computer-Aided Design, vol. CAD-4, no. 1, pp. 92-98, Jan. 1985.

[13] Analysis of Convergence Properties of a Stochastic Evolution Algorithm, Chi-Yu Mao and Yu Hen Hu, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 15, no. 7, Jul 1996

[14] http://vlsicad.eecs.umich.edu/BK/FEATURE/

[15] http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Placement/TESTCASES/

[16] Combinatorial Optimization by Stochastic Evolution, IEEE Trans. Computer-Aided Design, vol. 10. no. 4, Apr 1991

[17] Ant Colony Optimization, Marco Dorigo and Thomas Stutzle, ISBN 0262042193, Published 2004, MIT Press