A Greedy-Simulated Annealing approach for placement of VLSI circuits

Mahmoud Ahmed Sebak
Computer Engineering Dept.
Cairo University
Cairo, Egypt

Mazen Amr Fawzy
Computer Engineering Dept.
Cairo University
Cairo, Egypt

Mohamed Ahmed Ibrahim
Computer Engineering Dept.
Cairo University
Cairo, Egypt

Reham Gamal Elsaid
Computer Engineering Dept.
Cairo University
Cairo, Egypt

***Abstract*—In this paper, we propose a heuristic approach for finding near optimal placement of VLSI circuits. Our algorithm starts with a greedy approach for providing an initial placement before starting the iterative simulated annealing improvement. Experiments have been carried out and the results shows that the proposed solution is computationally less expensive than the simulated annealing and gives better placement solutions than the greedy algorithms.**

I. Introduction

The objectives of a placement algorithm are to get the minimum total area of the chip and the minimum total wire length of the nets. This requires optimizing chip area's usage to apply more functionality in the smallest possible area. This helps in getting the minimum wire length to reduce the capacitive delays for long nets and to increase the speed of operation of the chip. All These goals are related to each other for gate arrays and standard cells. But there is a trade-off between minimizing the wire length and minimizing the chip area. In some cases such as the preferential minimization of wire length of a few critical nets increase the total wire length.

There is another rule for getting acceptable placement algorithms is that it must be physically possible and this can be achieved by checking the following:

- Standard cells should be confined to rows in predetermined positions.
- No overlapping between the modules.
- They shouldn't exit or exceed the chips boundaries.
- Gates in a gate array should be confined to grid points.

It is preferred to define a cost function which is the sum of the total required wire length and conditions for module overlap, total required chip area, and any other rules or requirements. This will help in achieving and checking the goal of the placement algorithm in minimizing possible cost.

VLSI cell placement problem is considered NP complete problem so it cannot be solved exactly in polynomial time. It would take time proportional to factorial of the number of modules to try and get an exact solution by evaluating every possible placement to determine the best one. The placement process is followed by routing, that is to say determining the interconnects' physical layout through the available space. Finding an optimal routing given a placement is also NP complete problem. Many algorithms work by improving the placement iteratively, and estimating the wire length of an intermediate configuration at each step.

It is possible to divide the placement algorithms into two major classes: iterative improvement and constructive placement.

Iterative improvement algorithms start with an initial placement and modify it repeatedly searching for a cost reduction. If a modification results in cost reduction, the amendment will be accepted; otherwise it is rejected. Iterative enhancement algorithms generate typically good placements but require huge amounts of computation time.

The simplest strategy for iterative improvement interchanges, is randomly selecting pairs of modules, and accepting the interchange if it results in cost reduction. When there is no further improvement during a given large number of trials, the algorithm terminates.

Repeated iterative improvement is an improvement over this algorithm, in which the iterative improvement process is repeated several times with different initial configurations hoping to get a good configuration in one of the trials. Currently popular iterative enhancement algorithms include simulated annealing, genetic algorithm, and certain force-directed placement techniques.

Constructive placement is to build a placement from scratch. Constructive placement algorithms are based on primitive connectivity rules when selecting and placing a seed module in the chip layout area. Then, in order to connect it to the placed modules (most densely connected first) other modules are selected one at a time and placed at a vacant location close to the placed modules, so that the wire length is minimized. Such algorithms are usually very quick, but typically lead to poor layouts. These algorithms are now used to generate an initial placement for iterative improvement algorithms. Their usage is mainly due to their speed.

Comparing constructive algorithms to iterative improvement algorithms they take a negligible amount of computation time and provide a good starting point for them. More recent constructive placement algorithms, such as numerical optimization techniques, partitioning placement, and force driven techniques, yield better layouts but require significantly more CPU time.

II. Related work

A. Simulated annealing

Its one of the most well-developed algorithms used for placement of the modules. It can be considered an improved version of the simple random pairwise interchange algorithm.
Advantages: Its excellent results where it is an excellent heuristic for solving any optimization problem.
Disadvantages: It consumes a lot of time as it is an iterative method.

The basic procedure for simulated annealing as discussed in [1] is to accept all moves which lead to cost reduction. Moves resulting in cost increase are accepted with a likelihood that decreases with cost increase. A parameter T, called temperature, is used to control the likelihood of acceptance of the moves that increase the cost. Higher T values cause more acceptance of such moves. The acceptance probability in most implementations of this algorithm is given by $\exp(-\delta C/T)$, where δC is the increase in cost. The temperature is set to a very high value at the beginning, so that most moves are accepted. Then the temperature is gradually lowered so that moves that cause cost increases have less chance of being accepted. Ultimately, the temperature is reduced to a very low value so that only moves causing a cost reduction are accepted, and the algorithm converges to a low cost configuration.
This algorithm depends on initial random placing for the modules then starting of interchanging two modules, mirroring and rotation applying the layout geometry restrictions, or any other changes(moves) that make changes for the length of the wire.
It has an Inner loop criterion which determines the number of trials for each temperature. Usually the number of moves for each cell at each temperature is fixed. For macro blocks rotating in steps of 90 degree or mirroring about the two axis is allowed and mirroring around the vertical axis is only allowed for the standard cells. To save time of running CPU the wire length changes is calculated incrementally where the calculations is done only for the net that are connected to the one which is moving.

B. Force directed placement

It is one of the most important placement techniques proposed in 1967 [Fisk et al.] as it could be done in constructive manner, and iterative manner. The goal is to calculate the location where each module should be placed in order to achieve ideal placement.
Idea: as discussed in [1]. consider a given placement. Assume the modules that are connected to each other exert an attractive force on each other. To achieve ideal placement we should reach a point where the resultant of forces exerted on all modules is zero, in other words, equilibrium point, where all forces' magnitudes are proportional to the distance between modules, and the resultant of forces exerted on a module is the sum of them in vector form(magnitude,direction). We could model this idea as some blocks with springs in between, and the equilibrium point is where the tension in the springs is minimum, which is equivalent to minimum wire length.
Force-directed relaxation: a process in which we choose a module, calculate its target point then attempt to move it to the target point or interchange it with the module previously occupying it.
Goto's placement algorithm: as proposed in 1986 by [Goto and Matsuda]. It uses a force-directed iterative technique, and it consists of two parts. The initial placement part selects modules for placement based on connectivity. When a module is selected it is placed at a location that yields minimum wire length and not moved during the rest of the initial placement part. The iterative improvement part uses a force-directed relaxation technique in which interchange of two or more modules in the ϵ-neighborhood of the median of a module are explored to interchange with.
The median of a module is the position at which the wire length of the nets connected to the module is minimum.
The ϵ-neighborhood of a module is the set of ϵ positions in which the wire length between them and the selected module has the smallest ϵ values.
Finding the median and its ϵ neighborhood are separable in x and y, and hence x- and y-coordinates of the median can be calculated independently of each other using johnson and mizoguchi algorithm.

Before summarizing the steps we need to define the λ-neighborhood of a configuration in the configuration space as the set of configurations that can be obtained from the given configuration by circularly interchanging no more than λ modules. A configuration is called λ-optimal configuration if it is the best configuration in the λ-neighborhood. The process of replacing the current configuration with a better one from its λ-neighborhood is called local transformation.

The complete algorithm is summarized as follows: An initial placement is generated. An iterative improvement is performed as a generalized force-directed placement to obtain a λ-optimum configuration. If the computation time limit is not exhausted yet , then the procedure is repeated with another initial placement. The best result of all trials is taken. The only thing that remains is the procedure used for finding λ-optimum configurations, which consists of iterated module interchange cycles until no further improvement exists. At each cycle's beginning a seed module (M) is selected and interchanged on a trial basis with all modules in its ϵ-neighborhood M(i) where ($1 < \mathrm{I} < \epsilon$). in case of the existence of a reduction in wire length, the interchange that yields maximum reduction is taken, and the cycle terminates. If there is no reduction, a triple interchange is tried between the seed Module M, a module M(i) in its ϵ-neighborhood, and a module M(j) in the ϵ-neighborhood of M(i) ($1 < \mathrm{I,j} < \epsilon$).in case of the existence of a reduction in wire length, then the configuration that yields maximum reduction is taken. If there is no reduction, then the process is repeated with a quadruple interchange between modules and so on until we find a reduction. We can see that the complexity of this procedure depends on the configuration and increases exponentially in order of ϵ.

Advantages:

- Faster than simulated annealing.
- Good approximation of results.

Disadvantages:

- Still has high complexity.
- Lower solution quality compared to simulated annealing.
- Requires fine tuning of parameters (ϵ,λ).

C. Placement by partitioning

Placement by partitioning is one of the most commonly used placement algorithms based on recursively dividing a given cell into high densely sub cells such that the number of cuts is minimized. Each sub-cell is then assigned to one partition of the chip area.

Kernighan-Lin and Fiduccia-Mattheyses Algorithms: Most of the partitioning algorithms or min- cut algorithms are modified versions of Kernighan-Lin and Fiduccia-Mattheyses algorithms mentioned in [1].

Kernighan-Lin is a bi-partitioning algorithm that starts with two initial disjoints sets A and B of equal or nearly equal number of vertices. Then it searches for a sequence of swaps of a vertex from set A with another one from set B such that it maximizes the partition of these two sets and repeats until there is no swaps can be made. Local bad swaps can be made in order to achieve global better solution. The benefit of this algorithm over local search heuristics is that it allows the swap only if it improves the current cut. The best swap pair can be found in $O(n)$ where n is the the number of vertices in the graph and $O(d)$ is needed to update the data structures (priority queue) where d is the sum of the degree of the two vertices to be swapped. The overall complexity of a single iteration is $O(n^2 log(n))$.

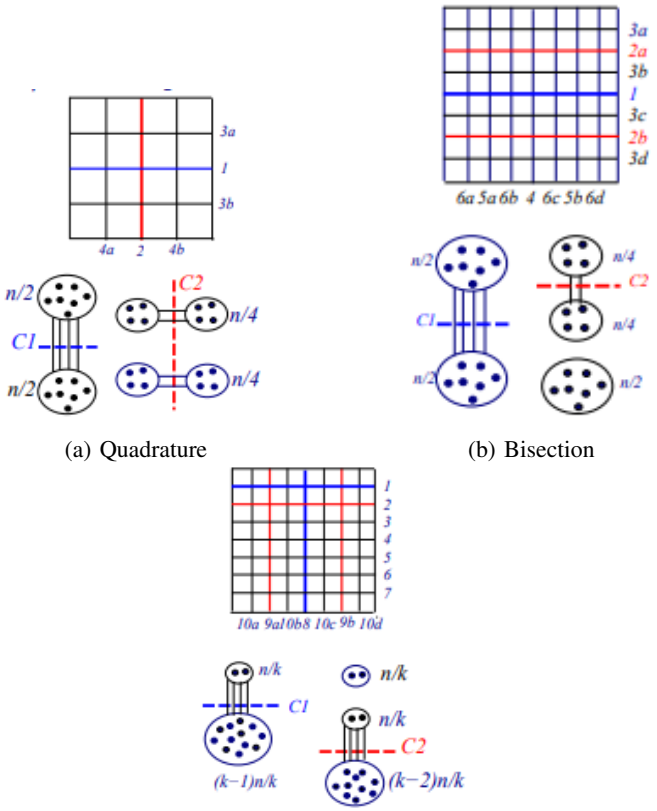
A much cheaper approach in terms of complexity was introduced by Fiduccia and Mattheyses which instead of swaps it makes vertex moves using special data structure in order to improve the running time. This approach takes into consideration the size of the chip. So, The number of the elements is each cell is not enough. The overall complexity of a single iteration is $O(n)$. However, it was proven that it is less powerful than the original Kernighan–Lin approach [4].

Breuer's Algorithms: Breuer's algorithms aims to minimize the number of partitions when the circuit is repeatedly cut by a set of cut lines. Breuer introduced two placement algorithms. Each of them requires cut lines to partition the chip into blocks.

a) Cut Oriented Min-Cut Placement Algorithms: Starting with the chip and a set of lines to partition the chip. The first line cuts the chip into two blocks and the circuit is divided among these two blocks into two sub-circuits minimizing the net cut. The blocks intersected by the second cutting line is then partitioned as well. Repeating this process for all the given cutting lines.

Disadvantages:

- Does not always reach global optimal placement.
- Sub-circuits may need larger space than the the assigned blocks.



(a) Quadrature (b) Bisection

(c) Slice Bisection

Fig. 1: Most popular sequences of cut lines for min-cut algorithms.

b) Block Oriented Min-Cut Placement Algorithms: This algorithm solves the problems of the previous Cut oriented algorithm as separate lines are used to partition every block until each block consists of one slot only. Different regions have different cutting lines instead of uniform cuts through the chip.

Partitioning procedures:

- **Quadrature Placement Procedure:** partitioning is done breadth first alternating vertical and horizontal cuts as shown in Fig.1a. This method is convenient for circuits having high density at the center.
- **Bisection Placement Procedure:** In this method partitioning is carried out by horizontal cutting lines until each region consists of one row. Then each row is bisected until each sub-region consists of a single slot and all modules are placed as shown in Fig.1b. This method is suitable for standard cell placement.
- **Slice Bisection Placement Procedure:** Another procedure is by partitioning the chip by horizontal cuts. Then assigning modules to each row. The modules in each row are assigned to columns by bisecting using vertical cutting lines as shown in Fig.1c. This method is suitable for cells with high interconnection on the periphery.

D. Guided Local Search

Guided local search [2] was proposed as an alternative iterative technique for simulated annealing due to it's high complexity in large circuits to reach the most optimized solution, it proposes combining **Guided Local Search (GLS)** with **Fast Local Search (FLS)** so, it focuses the search on appropriate sub-neighborhoods to reduce the time complexity relative to other iterative techniques.
Unlike other techniques this algorithm takes timing issues into consideration, since minimizing total length may leave critical nets having a significant signal delay.
The algorithm is starting by applying local search and start from initial solution χ_0 where $\chi_0 \in \chi$,and χ is a set of solutions that local search visits χ =χ_0,χ_1,χ_2,..,χ_k where each χ_i fulfills f(χ_0) > f(χ_1) > f(χ_2) .. > f(χ_k) where f(χ_i) are local optimization function so the loop ends when there is no solution better than χ_k solution.
GLS extends local search with the concept of features where these features characterize a solution to the problem we define. This is done to give high priority to attributes we need as area and wire length to decrease the search time. Since GLS is still time extensive, FLS is applied which divides the neighborhood into sub-neighborhoods which can be either active or not. At first all sub-neighborhoods are active. FLS visits the active sub-neighborhoods with any order. If a sub-neighborhood is examined and does not contain any improving move it becomes inactive. Otherwise it remains active and the improving move is performed, this is continuously done until all sub-neighborhoods are inactive (i.e. No more optimization available).

III. Proposed solution

Our proposed solution aims to reduce the total wire length by choosing where to place modules relative to each other. It consists of two stages.
Greedy stage: which aims to make an initial placement for the modules in our net list and outputs a matrix in which the relative positions of modules exist to reduce the complexity, or the taken time to complete the second stage.
Iterative stage: which improves the initial placement from the Greedy stage using simulated annealing. In this part, actions as moving a random module in any of the four directions or swapping it with a neighbouring module are performed until it converges to a global optimal placement.

A. Greedy stage

As shown in Algorithm 1 the steps are as follows:
We construct a weighted connectivity graph which indicates the number of connections between a module and another. A sample graph would look like Fig. 2.

Algorithm 1 Greedy stage

```
Construct weighted connectivity graph.
Construct closest neighbor graph.
Initialize Matrix.
Sort modules according to number of closest neighbors.
for i in 1 : n do
  if module is not visited then
    DFS(middle cell , current module)
  else
    DFS(closest cell to current module , current module)
  end if
end for
```

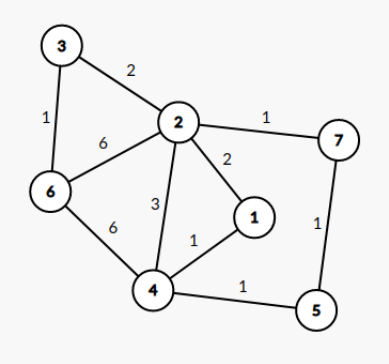


Fig. 2: weighted connectivity graph

Then we take the WCG graph and construct a closest neighbor graph, which indicates for each module the closest neighbor to it based on the strength of connection between them. The graph would look like Fig. 3.

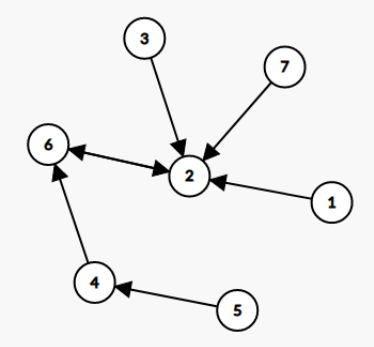


Fig. 3: closest neighbor graph

After that we initialize an empty square matrix with length $l = ceil(sqrt(n))$. where n is the number of modules.
Then given the closest neighbor graph we sort the modules based on how frequently it was the closest neighbor to other modules. Then we loop on all modules with the order we got and check if the module was placed earlier or not. If it was placed earlier then we pass it, if not then we check again if it was the first module to place or not. If it was the first then we start placing from the middle cell in the grid, if not then we

check if it has a neighbor that has been placed before to get the closest empty cell to it, if no neighbor module is found then we get the closest empty cell to the middle cell.
After determining the start cell, we start placing the modules in the matrix from the start cell using a depth first search.
We place neighbor modules in descending order of connectivity and so on until we process all modules. Modules are placed in counter clock wise direction of a 4-neighborhood starting from the bottom cell then the rest of cells in the 8-neighborhood starting from the bottom right cell. So the order of placing cells is (bottom cell , right cell , upper cell , left cell , bottom right cell , upper right cell , upper left cell , bottom left cell).

B. Iterative stage

After the initial placement is done by the greedy algorithm, the placement is improved using simulated annealing. As shown in Algorithm 2, we define an initial temperature of 1 and a minimum temperature of 10^{-4}. The current temperature is multiplied by a cooling factor, alpha equals 0.9 until it reaches the minimum temperature. For every temperature value, a fixed number of swaps and moves are made. Cost is then calculated for every action using Manhattan distance. Actions that result in a decrease in the cost will be accepted directly. However, actions that increases the cost will be accepted by a certain probability that depends on the current state (temperature) in an attempt to escape local optimal solutions and converge to a value near the global optimal solution.

Algorithm 2 Simulated Annealing

$T \leftarrow 1$
$Tmin \leftarrow 0.0001$
$alpha \leftarrow 0.9$
$iterations \leftarrow 10$
while $T >$Tmin **do**
  for i in $0 : iterations$ **do**
    $cost_{current} \leftarrow calculateCost(grid)$
    Make a move or swap and update grid
    $cost_{next} \leftarrow calculateCost(newGrid)$
    $diff \leftarrow cost_{current} - cost_{next}$
    if $diff > 0$ **then**
      Accept action made
    else
      $r \leftarrow random(0, 1)$
      if $r < \exp(-diff/T)$ **then**
        Accept action made
      end if
    end if
  end for
  $T \leftarrow$T * $alpha$
end while

IV. Comparison Results

The results in Table I. shows a comparison between the total wiring length of our proposed solution and another algorithm introduced in [5] that uses Eigen vectors to make an initial placement before starting the simulated annealing for different number of modules. The proposed greedy approach results in a better initial placement and gets the simulated annealing to reach a near optimal solution faster than using Eigen vectors and gives better results specially for large number of modules.

TABLE I: Eigen Vector vs Greedy approach

Modules	Eigen	Greedy	SA with Eigen	SA with greedy
20	112	62	58	56
30	412	178	146	148

Fig. 4. shows the results of an experiment. In this experiment, we create a number of modules with randomly generated connections. A grid is made randomly from these modules and the initial cost is calculated. Then, the cost after running simulated annealing directly is calculated and compared with the cost after using our greedy algorithm to make an initial placement before running the simulated annealing. The results shows that the greedy algorithm gives results nearly equals the results of simulated annealing.The proposed algorithm gives slightly better results specially for large number of modules.

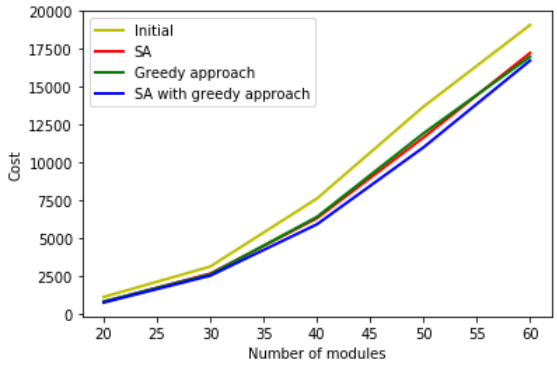


Fig. 4: Number of modules vs Cost graph

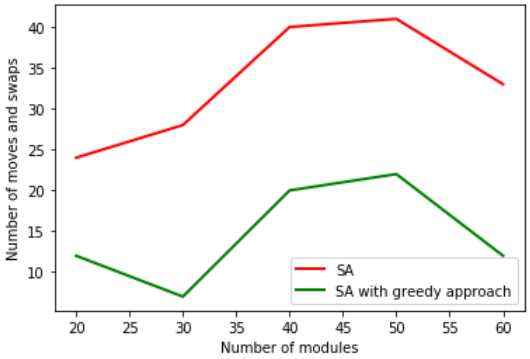


Fig. 5: Number of modules vs Number of moves and swaps graph

The graph shown in Fig. 5. shows a comparison between number of moves and swaps between Simulating annealing

and the proposed solution for different number of modules. The graph clearly shows that using greedy algorithm to give initial solution before simulated annealing reduces the number of swaps and moves needed and makes it converge faster towards the global optimal solution.

V. Conclusion and Future work

In this paper, we proposed a hybrid placement algorithm that retains the fast performance of a greedy algorithm and yields near optimal solutions using simulated annealing. Experimental studies are made to test the performance of the proposed algorithm. The results agree with the expected performance of having less wire length and requires less number of moves and swaps to reach near optimal solutions.

Our work will be directed towards exploring more greedy algorithms and apply them before different iterative algorithms including min-cut partitioning in order to enhance the performance and achieve better results.

References

[1] K.Shahookar and P.Mazumder,"VLSI cell placement techniques," ACM Computing Surveys, 1991.

[2] Faroe, Oluf and Pisinger, David and Zachariasen, Martin. (2003). "Guided Local Search for Final Placement in VLSI Design," Journal of heuristics, 2003.

[3] Mall, Rajib, and Lalit M. Patnaik, "A Force Directed Hill-Climbing Placement Algorithm," The Fifth International Conference on VLSI Design, IEEE, 1992.

[4] Träff, Jesper Larsson, "Direct graph k-partitioning with a Kernighan–Lin like heuristic," Operations Research Letters, 2006.

[5] Kymry Burwell, "VLSI Cell Placement" Github, 17 December 2019, www.github.com/kymry/VLSI-Cell-Placement/blob/master/.