Research Article

A Swarm Random Walk Based Method for the Standard Cell Placement Problem

Najwa Altwaijry and Mohamed El Bachir Menai

*Department of Computer Science, College of Computer and Information Sciences, King Saud University,
P.O. Box 51178, Riyadh 11453, Saudi Arabia*

Correspondence should be addressed to Najwa Altwaijry; ntwaijry@ksu.edu.sa

Received 12 March 2014; Accepted 19 July 2014; Published 13 August 2014

Academic Editor: Baozhen Yao

Copyright © 2014 N. Altwaijry and M. E. B. Menai. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The standard cell placement (SCP) problem is a well-studied placement problem, as it is an important step in the VLSI design process. In SCP, cells are placed on chip to optimize some objectives, such as wirelength or area. The SCP problem is solved using mainly four basic methods: simulated annealing, quadratic placement, min-cut placement, and force-directed placement. These methods are adequate for small chip sizes. Nowadays, chip sizes are very large, and hence, hybrid methods are employed to solve the SCP problem instead of the original methods by themselves. This paper presents a new hybrid method for the SCP problem using a swarm intelligence-based (SI) method, called SwarmRW (swarm random walk), on top of a min-cut based partitioner. The resulting placer, called sPL (swarm placer), was tested on the PEKU benchmark suite and compared with several related placers. The obtained results demonstrate the effectiveness of the proposed approach and show that sPL can achieve competitive performance.

1. Introduction

The placement problem has been studied extensively in the past decades, and it is a crucial problem in VLSI computer aided design. The object of this study is standard cell placement, one of the methods of placement. Standard cells are logic modules that have a predesigned internal layout. Standard cells are of a fixed height and varying widths, because of the different functionalities of the modules, and are laid out in rows on chip. Figure 1 shows an example of a standard cell layout style with whitespace in between, shown in gray [1]. Logic inputs and outputs are available at pins (or terminals) along the top and bottom edges of the cells, and wires that connect standard cells and pads pass through the whitespace and the area between rows of standard cells.

Primary objectives when placing cells on chip include wirelength minimization, where wirelength is defined as the sum of the total estimated interconnection wirelengths, and area minimization. Other objectives such as power optimization and timing optimization are common. The SCP problem is defined as follows. Given an electrical circuit consisting of standard cells with predefined input and output terminals and interconnected in a predefined way, construct

a layout indicating the positions of the cells so that objectives are optimized. The inputs to the problem are the descriptions of the standard cells and the netlist. The standard cell description consists of the shapes, sizes, and terminal locations. The netlist describes the interconnections between the terminals of the cells. The output is a list of x - and y -coordinates for all standard cells. More formally, it can be stated as follows. Given a set of n standard cells $C = \{c_1, c_2, \dots, c_n\}$, a set of k signals $S = \{s_1, s_2, \dots, s_k\}$, where each signal s_i is associated with a subset of cells C_{s_i} , where $C_{s_i} = \{c_j \mid s_i \in S\}$, and a set of locations $L = \{l_1, l_2, \dots, l_p\}$, where $p \geq n$. The SCP problem asks to assign each cell $c_i \in C$ to a unique location l_j , such that the objectives are minimized (or maximized) subject to constraints. Our chosen objective is to minimize the total wirelength TWL:

$$\text{TWL} = \sum_{i=1}^k \text{length}(s_i), \quad (1)$$

where $\text{length}(s_i)$ is the length of the net associated with signal s_i .

The SCP problem is an NP-hard problem [1, 2]. Current chip sizes may contain millions of movable objects, and,

FIGURE 1: Standard cell layout style.

as such, the problem cannot be solved exactly in polynomial time. For that reason, a metaheuristic algorithm is used to search the large solution space of the problem. Previously, methods used to solve the SCP problem were broadly categorized into iterative placement methods and constructive placement methods. Iterative placement methods can rarely create a good placement from a randomly generated starting placement, especially in chips with large numbers of cells, as well as having runtime requirements significantly larger than those required by constructive placement methods. Iterative placement methods include the simulated annealing and the genetic algorithm. Constructive placement algorithms generally lack accuracy on placement objectives such as wirelength [3], and they include the min-cut algorithm, the force-directed algorithm, and the quadratic placement algorithm. A combined approach is more likely to yield a successful placement [3].

In this paper, we propose a novel hybrid method to solve the SCP problem based on the algorithm swarm random walk (SwarmRW) [4] together with hMetis, a min-cut based partitioner [5, 6], resulting in a placer called sPL (swarm placer). The objective function chosen in this work is wirelength minimization. In some cases, secondary performance measures are adopted, such as minimizing the length of some critical nets. The cost of total wirelength may increase, as these objectives are generally conflicting. Other objectives include power consumption and routability. Weighted total wirelength is a convenient representative because it can be optimized efficiently, and with some net reweighting, it can be used to represent other objectives [7].

Wirelength is estimated by the semiperimeter method, in which the half-perimeter of the bounding rectangle of all pins in the net is taken to be the wirelength of the net. Experiments were conducted on the PEKU benchmark [8], which contains several sets of placement examples with known upper bound of wirelength, containing local nets as well as nonlocal nets. sPL was compared against other placers including Dragon

[9], a proven and well-known placer based on partitioning and simulated annealing, and abcPL [10], a placer based on an artificial bee colony (ABC) algorithm [11]. The results obtained demonstrate the competitiveness of the proposed approach and show particularly that sPL outperforms abcPL.

The rest of this paper is organized as follows. Section 2 reviews some of the most prominent works in cell placement. Section 3 describes the general SwarmRW algorithm. Section 4 describes the sPL placer. Section 5 shows the experimental results obtained by sPL and discusses its performance compared with others. Finally, Section 6 concludes this work and outlines some future research directions.

2. Related Work

As mentioned previously, SCP algorithms can be broadly categorized into constructive placement methods and iterative placement methods. These methods, by themselves, are rarely used today. Instead, hybridization of these methods, with some advanced heuristics, is used to solve the SCP problem.

QPlace [12] is an industrial placer by Cadence Design Systems, Inc., but a detailed description is unavailable to the authors as it has been discontinued. mPG [13] is a multilevel and simulated annealing placement algorithm with integrated global routing to update and optimize the cost of congestion. Global routing is done via a fast routing and incremental A-tree algorithm. Congestion is highlighted because overly congested areas force connections to make detours or to change layers, thereby increasing the actual wirelength and reducing performance.

mPL [14, 15] is an efficient multilevel placer that uses nonlinear programming to minimize quadratic wirelength. It uses recursive first choice clustering in the coarsening phase. A custom nonlinear programming solver using the interior point method is used at mPL's coarsest level to obtain an initial solution. Relaxation is restricted to sweeps of local refinements and spreading out of cells.

Dragon [9, 16] is a top-down hierarchical placement tool that deploys min-cut based partitioning and simulated annealing to place designs with thousands of macroblocks and millions of standard cells. Dragon emphasizes congestion during placement, in addition to traditional wirelength minimization. Congestion is a main objective during routing; highly congested regions necessitate routing detours around the site, leading to increased wirelength, and in the worst case, the placement is unroutable. In order to reduce congestion, it can be included in the cost function of simulated annealing, by combining a regional router into the placement, or in a postprocessing step. Dragon uses white space to improve routability dynamically during placement. In addition, two methods to control target utilization are incorporated into Dragon. Target utilization is a metric that estimates the capability of the placement tool to generate routable designs.

CAPO [17] is a placer that implements a top-down cut-size driven recursive bipartitioning and uses the multilevel hypergraph partitioner MLpart [18] and some time-limited branch-and-bound heuristics. It has some enhancements to avoid the corking effects caused by balance constraints, as well as terminal propagation.

abcPL [10] is a top-down hierarchical placer that uses hMetis [5, 6], a min-cut based partitioner, and the ABC algorithm [11] to place a chip, minimizing wirelength. These methods have all been tested on the PEKU benchmark suite [8], and the wirelengths produced can be 2.98 times the optimal in the worst case.

There remains significant room for improvement in existing placement algorithms, suggesting that new—more scalable and stable—hybrid techniques may be needed for future generations. There are other methods that have been used to solve the SCP problem, using genetic algorithms (parallel genetic algorithm (PGA) [19] and genetic algorithm for placement (GAP) [20]), simulated evolution (force-directed simulated evolution [21] and distributed parallelized SE algorithm (SimE) [22]) in addition to some hybridized methods (parallel simulated annealing/genetic algorithm (PSAGA) [23] and SimE-GA [24]); however, these methods are not as competitive, being tested on older benchmarks. Although simulated evolution was used to solve instances of the PEKU suite in [25], it is not included in the experimental results as it was uncompetitive, with quality ratios ranging from 6.33 to 8.52.

3. SwarmRW Algorithm

SwarmRW [4] is a new SI algorithm. It employs a swarm of potential solutions that cooperate or “learn” from each other one dimension at a time, to optimize each swarm member. First, a swarm of initial solutions is randomly generated, and each individual then generates a new solution. Individuals learn from each other the best values, or positions, and change themselves accordingly: each individual will “imitate” a randomly selected neighbor in a single randomly selected dimension at a time, and if this improves the fitness of the individual, the change will be accepted; otherwise, it will be rejected. Intuitively, changing the solution in a single dimension at a time allows the search process to better judge if the proposed change is beneficial.

For an optimization problem with d dimensions, a swarm of n initial solutions $\mathbf{x}_i = (x_{i1}, \dots, x_{id})$, $1 \leq i \leq n$, is randomly generated. A change to $\mathbf{x}_i$ is made in a single dimension j , and a greedy selection is performed. If this move results in a better fitness value, the change is accepted. However, if the move decreases fitness, the move is rejected. The change made to x_{ij} is done through swarm cooperation. A random solution $\mathbf{x}_k$, $1 \leq k \leq n$ and $k \neq i$, is selected, and solution $\mathbf{x}_i$ imitates solution $\mathbf{x}_k$ in dimension j .

SwarmRW changes a solution in only one dimension j at a time t , using a simple formula:

$$x_l^{t+1} = \begin{cases} x_l^t + \rho * (x_l^t - y_l^t), & l = j, \\ x_l^t, & l \neq j, \end{cases} \quad (2)$$

where j is a randomly selected dimension, and $\mathbf{y}$ is a randomly selected solution, distinct from $\mathbf{x}$. The parameter ρ is a randomly generated number that describes how much $\mathbf{x}$ is attracted or repelled by $\mathbf{y}$. This process is a random walk performed by the swarm. Algorithm 1 outlines the main steps of SwarmRW [4].

The SwarmRW method is similar to how employed bees in the ABC algorithm produce neighboring solutions. However, the main difference between SwarmRW and ABC is that no onlooker bees are employed to probabilistically improve the best solutions produced by employed bees. In addition, solutions are never abandoned and scout bees are not used to replace abandoned solutions with new solutions in the search space.

4. sPL Placer

Simulated annealing (SA) has been heavily researched for SCP and has proven to be successful (e.g., Dragon [16]). Dragon replaces the current solution by a random neighboring state solution constructed from the current solution by moving a randomly selected cell or groups of cells to a new position. The neighboring solution may be accepted or rejected with a probability that depends on the change in the objective function value and the temperature, T . The temperature is gradually decreased throughout the SA process. When T is large, the probability of accepting a solution that decreases the objective function value is high, allowing the solution to move out of local optima. As T decreases, the acceptance probability goes down, meaning that the probability of accepting inferior solutions decreases, allowing the algorithm to converge to a close-to-optimal solution. SwarmRW works similarly by moving cells or groups of cells to new positions; however, it does not employ a temperature to accept inferior solutions with a certain probability and only greedily accepts solutions that improve current solution quality. It is our belief that SwarmRW would be equally applicable to problems other than SCP, with no added benefit from its properties that are specific to the SCP problem.

sPL solves large-sized SCP problems with the objective of minimizing wirelength. sPL is implemented in a top-down hierarchical approach. While hMetis partitions cells into bins using recursive bisection with the objective of minimizing the net-cut, SwarmRW uses the wirelength objective (half-perimeter wirelength) to place the bins. Each swarm member (potential solution) contains the following information: the number of bins and their arrangement on chip, the total wirelength of the bins, in addition to the net length of each net, and the number of rows and columns on chip.

Initialization is performed as follows: an initial partition of 64 bins is created using hMetis, where half the bins are initialized randomly, while the rest are initialized sequentially (see Figures 2 and 3). Bins are of equal size and contain approximately the same number of cells, with each bin having a unique numerical identifier. Cells within the bins are placed on top of each other in the lower left corner of the bin (i.e., cells overlap in bins). Optimization of bin positions commences through SwarmRW. After a preset maximum number of iterations, or if no reduction in wirelength is noted for some iterations, SwarmRW relinquishes the bins to hMetis, which partitions them either horizontally or vertically, in turn.

The SwarmRW algorithm is converted to solve the discrete SCP problem, as it was designed for continuous domain

```

input:  $S, \max, \rho$ 
output: Best Solution
Initialize:  $S$ : swarm size,  $\max$ : maximum runs,  $\rho$ : parameter;
begin
  Initialize swarm randomly;
  for  $\text{iter} = 0 \rightarrow \max - 1$  do
    for  $i = 0 \rightarrow S - 1$  do
      Randomly select  $k, 0 \leq k < S, k \neq i$ ;
      Set  $\mathbf{y} = \mathbf{x}_k$ ;
      Randomly select dimension  $j$ ;
      Calculate  $x_{ij}^{t+1}$  by (2);
      Set  $\bar{\mathbf{x}}_i = (x_{i1}^t, \dots, x_{ij}^{t+1}, \dots, x_{id}^t)$ ;
      if  $\text{fitness}(\mathbf{x}_i) < \text{fitness}(\bar{\mathbf{x}}_i)$  then
         $\mathbf{x}_i = \bar{\mathbf{x}}_i$ ;
      end
    end
  end
end
return Best Solution

```

ALGORITHM 1: SwarmRW pseudocode.

14	10	6	2
11	7	3	15
8	4	0	12
5	1	13	9

FIGURE 2: Random initialization of bins.

3	7	11	15
2	6	10	14
1	5	9	13
0	4	8	12

FIGURE 3: Sequential initialization of bins.

problems. For example, the PSO algorithm can be converted from its continuous version into a discrete version to solve the TSP problem by using a series of swaps (swapping cities) instead of a velocity. A similar method is adopted for SwarmRW, where only a single swap can be made at a time, in keeping with updating only a single dimension at a time in the continuous version. An individual in SwarmRW will produce a nearby solution by swapping two bins, and greedy

selection is used to select the best solution from among the new and old solutions.

Bins are partitioned at the end of each SwarmRW phase, either vertically or horizontally. If the last cut was vertical, the bins are partitioned horizontally, and vice versa. The cells and nets within the bin are fed into hMetis using the appropriate format, disregarding connections to outside bins. The resulting partitions are then used to continue the optimization process.

This two-step process is repeated until each bin has 3-4 cells [9, 26]. Finally, at the end of a placement run, the bin structure is discarded, and bins are converted into rows, where the cells in each bin are placed within the bin area on chip. While placing cells, the chip width and the number of rows are taken into consideration. Columns of bins might contain a number of cells larger than the available space on chip for those columns, in which case cells are placed as close to their intended position as possible. Finally, a greedy heuristic further improves wirelength. Pseudocode for our SPL placer is shown in Algorithm 2.

4.1. Fitness Function. The fitness function uses the semiperimeter method for calculating wirelength and is calculated as follows:

$$\text{fitness} = \frac{10,000,000}{\text{wirelength} + 1}, \quad \text{wirelength} \geq 0, \quad (3)$$

where the wirelength is calculated as shown in (4) and net_i^u is calculated as in (5). Consider the following:

$$\text{wirelength} = \sum_{i=0}^{\max_{\text{nets}}-1} \text{net}_i^u, \quad (4)$$

$$\text{net}_i^u = ((X_{\max} - X_{\min}) + (Y_{\max} - Y_{\min})) + (\text{cells}_i * w), \quad (5)$$

```

input:  $S$ ,  $\max$ , StoppingCondition, benchmark files
output: Best Placement
Initialize:  $S$ : swarm size,  $\max$ : maximum runs;
begin
  Read benchmark files;
  repeat
    Partition Bins;
    Initialize swarm;
    for  $\text{count} = 0 \rightarrow \max - 1$  do
      for  $i = 0 \rightarrow S - 1$  do
        Generate a solution  $\bar{x}_i$  using a method from Section 4.2;
        if  $\text{fitness}(\mathbf{x}_i) < \text{fitness}(\bar{\mathbf{x}}_i)$  then
           $\mathbf{x}_i = \bar{\mathbf{x}}_i$ ;
        end
      end
    end
  until StoppingCondition = true;
  Remove bin structure to convert best solution  $\mathbf{x}$  bins to rows;
  Use greedy heuristic;
  Output wirelength;
  return Best Placement
end

```

ALGORITHM 2: SPL pseudocode.

where net_i^u is the total wirelength (taking into consideration the number of cells in the net), $X_{\max}$ is the maximum x -coordinate of all cells in the net, $X_{\min}$ is the minimum x -coordinate, $Y_{\max}$ is the maximum y -coordinate, $Y_{\min}$ is the minimum y -coordinate of all cells in the net, cells_i is the number of cells in net_i^u , and w is the average cell width of the benchmark cells.

4.2. Producing Neighboring Solutions. An individual produces a nearby solution by swapping two bins, and greedy selection is used to select the best solution from among the new and old solutions. To produce a nearby solution, the following methods are implemented.

- (1) Switch one bin using a neighbor's content: a bin is moved to a new position in the individual, according to that bin's position in a neighboring individual.
- (2) Randomly select two bins and switch their places.
- (3) Switch a randomly selected bin with a randomly selected nearby bin. A nearby bin is defined as a bin that is above or below the selected bin by a preset number of spaces, usually less than 8.
- (4) Switch a randomly selected bin with a bin around its perimeter. Note that bins around the edge of the chip have fewer neighbors and thus fewer bins to be switched with.

The first method is the only one in which the swarm "cooperates" to produce better solutions. The other methods were added as they are acceptable variations of changing the solution in one dimension at a time. Note that other methods could have been considered, such as subtracting one solution from another and then selecting one of the different

swaps (needed to change one solution into another) produced by such a subtraction, to perform the swap. The SwarmRW parameter p can be used to affect the probability of choosing one of these swaps. However, method 1 above produces one of these differences randomly without needing to perform a lengthy subtraction. Calculating the entire difference between two individuals is needed for algorithms such as PSO, where the velocity is changed over all dimensions simultaneously; however, in SwarmRW's case, it is unnecessary.

4.3. Converting Bins to Rows. Usually, the number of bin rows does not correspond to the real number of rows on chip. In the final SPL phase, bins are converted into rows. A column of bins is merged and cells are placed within the column area. Cells are placed beginning with cells contained in the bottom leftmost bin. When the first row area is placed, and if there are some remaining cells in the bin, the remaining cells are placed in the next row (the row above). When all cells in a bin are placed, the bin above it in the same column is placed in the same manner, before placing the next column of bins. If the space available to a column of bins is used and there are some unplaced cells, they are placed in the nearest empty position. However, if the number of unplaced cells is above a certain threshold, the bin is allowed extra width on chip, to allow all cells to be placed within bin boundaries.

5. Experimental Results

sPL was implemented in Java. Some experiments were performed on an Intel Xeon 3.20GHzx4 CPU, with 32 GB RAM under 64-bit Ubuntu 12.04.1, and some were performed on Intel Core 2 Duo 2.80GHzx2 CPU, with 6.8 GB RAM under 64-bit Ubuntu 12.04.1. Two experiments were performed

FIGURE 4: Wirelength variation on PEKU with 0.0% nonlocal nets.

FIGURE 6: Wirelength variation on PEKU with 0.25% nonlocal nets.

FIGURE 5: Quality ratios on PEKU with 0.0% nonlocal nets.

FIGURE 7: Quality ratios on PEKU with 0.25% nonlocal nets.

on the Intel Xeon in tandem. The latest version of hMetis (2.0pre1), with support for 64-bit architectures, was used. The benchmark suite selected for this study is the PEKU suite [8]. PEKU contains several sets of placement examples with a known upper bound of wirelength, containing local nets as well as nonlocal nets. There are 5 basic chip layouts in PEKU, each having 8 different percentages of nonlocal nets, for a total of 40 instances. The number of movable objects ranges from tens of thousands to hundreds of thousands of cells. As an example, PEKU01 contains 12506 cells, 14111 nets, and 113 rows. The upper bound on the wirelength is 8:14e5 units.

sPL is compared with abcPL, as well as mPL, mPG, CAPO, and Dragon. Results achieved by these placers on the selected benchmark are reported in the literature. These placers are state-of-the-art academic placers and have achieved the best reported results on our chosen benchmark. The other methods reported in the related work are not as competitive, being tested on older benchmarks.

An initial tuning of the algorithm's parameters on PEKU01 showed that varying the swarm size from 3 to 10 resulted in an average increase of 3% in the performance of sPL. However, the required runtime also increased by nearly

FIGURE 8: Wirelength variation on PEKU with 0.50% nonlocal nets.

FIGURE 10: Wirelength variation on PEKU with 0.75% nonlocal nets.

FIGURE 9: Quality ratios on PEKU with 0.50% nonlocal nets.

FIGURE 11: Quality ratios on PEKU with 0.75% nonlocal nets.

334%. This prohibitive amount of runtime constrained us to use a colony size of 3. The maximum number of cycles allowed to improve individuals in the swarm is set to 70000. Individuals are optimized in each partitioning step for this maximum, and if no improvement is detected for a period of 10000 cycles, then an early exit strategy is enforced, moving the placement into the next partitioning phase. Partitioning is stopped when the number of cells in each bin is 3-4, as recommended in [9, 26].

Figure 4 shows the wirelength variation of the placers on PEKU with 0.0% nonlocal nets. Figure 5 shows the quality ratio of sPL on PEKU with 0.0% nonlocal nets. The quality

ratio is the wirelength achieved by the placer divided by the minimum wirelength. The results of sPL were averaged over 10 runs. QPlace [12] is the placement engine used in the Silicon Ensemble of Cadence. The version used is QPlace 5.1.55, in Silicon Ensemble v.5.3.

On PEKU with 0.25% to 10% nonlocal nets, sPL was run for a total of 5 times. Figures 6, 8, 10, 12, 14, 16, and 18 show the wirelengths. The quality ratios are reported in Figures 7, 9, 11, 13, 15, 17, and 19. The average runtimes for sPL and abcPL are shown in Figure 20.

The performance of sPL improves dramatically as the number of nonlocal nets increases, as shown in Figures 4,

FIGURE 12: Wirelength variation on PEKU with 1% nonlocal nets.

FIGURE 14: Wirelength variation on PEKU with 2% nonlocal nets.

FIGURE 13: Quality ratios on PEKU with 1% nonlocal nets.

FIGURE 15: Quality ratios on PEKU with 2% nonlocal nets.

6, 8, 10, 12, 14, 16, and 18 for the variation of the wirelength and in Figures 5, 7, 9, 11, 13, 15, 17, and 19 for the variation of the quality ratio. This trend is similar to the performance improvement of Dragon and abcPL. This might be a result of increased nonlocal nets providing more global information on the optimal positions of cells to sPL (and other placers), allowing them to better place these cells. Nonlocal nets offer more leeway for placement of the cells compared with just local nets, allowing deeper wells in the fitness function and lesser local minima. This might explain the placer's ability to achieve better solutions.

sPL shows that the performance of a random walk of a swarm is a promising approach for solving the SCP problem. The placer is not mature, and, as such, the performance of the first version is good compared to other, more mature, placers. Using SwarmRW is better than using ABC on all instances except for PEKU1 0% and 2%, where it is slightly worse. sPL is faster than abcPL on all tested instances. However, runtimes are long compared to mathematical tools, such as mPL. This can be explained by the frequent reads and writes to disk while partitioning using hMetis. sPL is better than mPL on all instances with 0.50% and over nonlocal nets. On instances with 0.75% and above, sPL performs comparably with mPG,

FIGURE 16: Wirelength variation on PEKU with 5% nonlocal nets.

FIGURE 18: Wirelength variation on PEKU with 10% nonlocal nets.

FIGURE 17: Quality ratios on PEKU with 5% nonlocal nets.

FIGURE 19: Quality ratios on PEKU with 10% nonlocal nets.

and outperforms it on some instances. It is comparable to Dragon and CAPO on instances with 5% and 10% nonlocal nets.

Results of sPL and the rival placers were analyzed using SPSS (<http://www-01.ibm.com/software/analytics/spss/>). The Friedman test was performed on the results obtained by the following placers: CAPO, Dragon, mPG, mPL, abcPL, and sPL. Results for PEKU 0.0% were not included in the test as mPG results are unavailable on those instances, so the total number of instances in the test is 35. Table 1 presents the descriptive statistics. The test showed there was a statistically

significant difference among all the placers with a P value of 0.000.

Next, the Wilcoxon ($1 * N$) signed-ranks test was run on sPL and the rest of the placers for pairwise analysis. Table 2 summarizes the results and the statistical significance among the placers. There is a statistically significant difference among all placer combinations except for mPG-sPL. Overall, sPL is better than mPL and abcPL is not statistically different from mPG and is worse than CAPO and Dragon.

We think that the performance of sPL can be improved by investigating other partitioning algorithms and heuristics to

FIGURE 20: Average runtimes of abcPL and sPL.

TABLE 1: Friedman test descriptive statistics.

Placer	Mean rank	Mean	Standard deviation
Dragon	1.63	$4.94e + 7$	$7.74E + 07$
CAPO	1.90	$5.21e + 7$	$8.34E + 07$
mPG	3.47	$5.81e + 7$	$9.24E + 07$
sPL	3.50	$5.47e + 7$	$8.01E + 07$
mPL	5.67	$8.36e + 7$	$1.22E + 08$
abcPL	4.83	$6.15e + 7$	$9.51E + 07$

TABLE 2: Wilcoxon test.

Placers	z-test	P value (2-tailed)	Rank
Dragon-sPL	-5.433	0.000	Positive
CAPO-sPL	-3.582	0.000	Positive
mPG-sPL	-0.402	0.688	Positive
mPL-sPL	-3.992	0.000	Negative
sPL-abcPL	-5.394	0.000	Positive

generate neighbors. sPL uses a heuristic to reduce wirelength after converting bins to rows and final cell placement on the chip. Other heuristics, such as the one in Dragon, may also be beneficial to the wirelength reduction objective. In addition, partitioning and then merging within the same level may allow better recovery from partitioning errors. This is because if a cell was incorrectly partitioned, merging before repartitioning may allow the cell to be placed in the correct partition.

6. Conclusion and Future Work

This paper has introduced a new placer called sPL that solves the SCP problem based on the SwarmRW metaheuristic and min-cut based partitioner hMetis. The experimental results obtained on the PEKU benchmark demonstrate that sPL

outperforms particularly abcPL, a metaheuristic based placer, and mPL, a nonlinear programming based placer, on most of the instances. Moreover, sPL's performance remains close to the performance of two other well-established placers: Dragon and CAPO. Therefore, this shows that SwarmRW is a promising approach for the SCP problem, particularly for large problem instances as the results of wirelength and quality ratios improve dramatically for sPL as the number of nonlocal nets increases.

Future work includes converting the algorithm to deal with macroblocks in addition to standard cells, in order to solve the mixed-size cell placement problem. Other flat partitioners may produce partitions more suited to our approach, as the metaheuristics alter the placement after partitioning. In addition, other methods for producing neighboring solutions can also be studied. Investigation of various heuristics to improve the solution might also be beneficial. In addition, investigating other versions of SwarmRW for the SCP by testing various escape strategies may result in improving the performance of the proposed placer.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This research project was supported by a grant from the Research Center of the Center for Female Scientific and Medical Colleges, Deanship of Scientific Research, King Saud University.

References

- [1] K. Shahookar and P. Mazumder, "VLSI cell placement techniques," *ACM Computing Surveys*, vol. 23, no. 2, pp. 143–220, 1991.
- [2] W. E. Donath, "Complexity theory and design automation," in *Proceedings of the 17th Design Automation Conference*, pp. 412–419, ACM, 1980.
- [3] M. Wang, X. Yang, and M. Sarrafzadeh, "Dragon2000: standard-cell placement tool for large industry circuits," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD '00)*, pp. 260–263, IEEE, 2000.
- [4] N. Altwaijry and M. B. Menai, "A swarm random walk algorithm for global continuous optimization," in *Proceedings of the 7th International Conference on Genetic and Evolutionary Computing (ICGEC '13)*, J.-S. Pan, Ed., vol. 238 of *Advances in Intelligent and Soft Computing*, pp. 33–43, Springer, 2014.
- [5] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: application in VLSI domain," in *Proceedings of the 34th Design Automation Conference*, pp. 526–529, June 1997.
- [6] N. Selvakkumaran, "Multiobjective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 3, pp. 504–517, 2006.
- [7] J. Cong, J. R. Shinnerl, M. Xie, T. Kong, and X. Yuan, "Large-scale circuit placement," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 2, pp. 389–430, 2005.

- [8] J. Cong, M. Romesis, and M. Xie, "Optimality, scalability and stability study of partitioning and placement algorithms," in *Proceedings of the International Symposium on Physical Design (ISPD '03)*, pp. 88–94, ACM, New York, NY, USA, April 2003.
- [9] X. Yang, *Congestion and timing optimization for standard-cell placement [Ph.D. thesis]*, University of California, Los Angeles, Calif, USA, 2002.
- [10] N. Altwaijry and M. B. Menai, "Artificial bee colony for the standard cell placement problem," *International Journal of Metaheuristics*, vol. 2, no. 3, pp. 234–255, 2013.
- [11] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Tech. Rep. tr06, Erciyes University Press, Erciyes, Turkey, 2005.
- [12] Cadence Design Systems Inc, "Envisia ultra placer reference," QPlace Version 5.1.55, 1999.
- [13] C. C. Chang, J. Cong, and Z. D. Pan, "Physical hierarchy generation with routing congestion control," in *Proceedings of the International Symposium on Physical Design (ISPD '02)*, pp. 36–41, Del Mar, Calif, USA, April 2002.
- [14] T. F. Chan, J. Cong, T. Kong, and J. R. Shinnerl, "Multilevel optimization for large-scale circuit placement," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 171–176, IEEE Press, 2000.
- [15] T. F. Chan, J. Cong, T. Kong, J. R. Shinnerl, and K. Sze, "An enhanced multilevel algorithm for circuit placement," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, p. 299, IEEE Computer Society, 2003.
- [16] T. Taghavi, X. Yang, B.-K. Choi, M. Wang, and M. Sarrafzadeh, "Congestion minimization in modern placement circuits," in *Modern Circuit Placement*, G. J. Nam and J. Cong, Eds., Integrated Circuits and Systems, pp. 135–163, Springer, 2007.
- [17] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can recursive bisection alone produce routable placements?" in *Proceedings of the 37th Design Automation Conference (DAC '00)*, pp. 477–482, ACM, June 2000.
- [18] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Improved algorithms for hypergraph bipartitioning," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 661–666, ACM, 2000.
- [19] P. Subbaraj, S. Saravana Sankar, and S. Anand, "Parallel genetic algorithm for VLSI standard cell placement," in *Proceedings of the International Conference on Advances in Computing, Control and Telecommunication Technologies (ACT '09)*, pp. 80–84, Trivandrum, India, December 2009.
- [20] T. W. Manikas and M. H. Mickle, "A genetic algorithm for mixed macro and standard cell placement," in *Proceedings of the 45th Midwest Symposium on Circuits and Systems (MWSCAS '02)*, vol. 2, pp. II-115–II-118, Tulsa, Okla, USA, August 2002.
- [21] J. A. Khan and S. M. Sait, "Fast fuzzy force-directed/simulated evolution metaheuristic for multiobjective vlsi cell placement," *Arabian Journal for Science and Engineering*, vol. 32, no. 2, pp. 263–280, 2007.
- [22] S. M. Sait, M. I. Ali, and A. M. Zaidi, "Multiobjective vlsi cell placement using distributed simulated evolution algorithm," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 6, pp. 6226–6229, Kobe, Japan, May 2005.
- [23] Z. Zha, *A highly scalable, parallel genetic algorithm for standard cell placement in VLSI design [Ph.D. thesis]*, Lamar University, Beaumont, Tex, USA, 2004.
- [24] S. M. Sait, M. Faheemuddin, M. I. Ali, and S. Sanaullah, "Simulated evolution based hybrids for genetic algorithm and tabu search," in *Proceedings of the International Conference on Computers and Their Applications (CATA '07)*, pp. 30–35, Honolulu, Hawaii, USA, March 2007.
- [25] Y. Shiraishi, T. Ono, and M. A. E. Dahb, "Solution space reduction of simulated evolution algorithm for solving standard cell placement problem," in *Proceedings of the 5th International Conference on Natural Computation (ICNC '09)*, pp. 420–424, Tianjin, China, August 2009.
- [26] X. Yang, M. Wang, K. Eguro, and M. Sarrafzadeh, "Snap-on placement tool," in *Proceedings of the International Symposium on Physical Design*, pp. 153–158, ACM, April 2000.