Ant Colony Optimization Technique for Macrocell Overlap Removal

Stelian Alupoaei[1]
University of South Florida
Dept. of Computer Science & Eng.
Tampa, FL, USA
alupoaei@eng.usf.edu

Srinivas Katkoori
University of South Florida
Dept. of Computer Science & Eng.
Tampa, FL, USA
katkoori@csee.usf.edu

Abstract

We present a novel macrocell overlap removal algorithm, based on the ant colony optimization (ACO) metaheuristic. The algorithm generates a feasible placement from a relative placement with overlaps produced by some placement algorithms such as quadratic programming and force-directed. It uses the concept of ant colonies, a set of agents that work together to improve an existing solution. Each ant in the colony will generate a placement based on the relative positions of the cells and feedback information about the best placements generated by previous colonies. The solution of each ant is improved by using a local optimization procedure.

1. Introduction

While several macrocell placement techniques, such as sequence pair [1], O-tree [2], and B*-tree [3], generate overlap free placements, others such as quadratic programming [4] and force-directed [5], generate layouts with cell overlaps. Allowing overlaps during the placement or floorplanning process can lead to better solution, but at the cost of a non feasible placement. The increase of the overlap cost at the end of the process may reduce the amount of overlap, but cannot eliminate it completely.

Vijayan and Tsay [6] proposed a method to remove the overlap based on the constraint graphs. The initial floorplan is used to build a set of constraint graphs with redundant edges. The graphs are topologically sorted and the critical (longest) path is determined for each of them. The algorithm removes a redundant edge from the most critical of the two critical paths. This step is repeated until there is no improvement in the layout area. The algorithm also uses a similar procedure to change the aspect ratio of the blocks. The disadvantage of this method is the random choice of the edge to be removed. If there are two or more edges which qualify for removal, it is not possible to predict which one will lead to a better final placement. Asato and Ali [7] proposed an improvement of this algorithm by removing all redundant edges from one of the constraint graphs after the iterative improvement proposed in [6] is finished.

Nag and Chaudhary [8] have used the sequence pair representation to remove the overlaps for FPGAs.

The rest of the paper is organized as follows: Section 2 presents the ACO heuristic. Section 3 describes the proposed procedure. Section 4 shows the method used to reduce the execution time. Section 5 presents and discusses the experimental results. Finally, Section 6 draws conclusions.

2. Ant Based Optimization Metaheuristic

Ant colony optimization (ACO) was introduced by Dorigo *et al.* [9] for solving the Traveling Salesman Problem (TSP). Consider a complete undirected graph $G(V, E)$, where each edge $(u, v) \in E$ has an associated nonnegative weight $w(u, v)$. TSP is defined as the problem of finding a minimal weight tour, i.e., a simple cycle which contains all vertices. The virtual ants have the same behavior as the real ants: they traverse the graph and lay pheromone trails on edges. At each vertex, they choose the next path based on the pheromone intensity trails of the edges incident in the vertex and the weight of the edges. Initially, m ants are placed in randomly selected vertices. Each ant will generate a tour and will deposit on the edges of the tour a pheromone trail proportional to the length of the tour.

Let $\tau(r, s; t)$ be the intensity of the trail on edge (r, s) at time t. After all ants have generated the tours, the trail intensity is updated (due to evaporation and trail reinforcement)

1 Stelian Alupoaei is now with Intel Corp., Hillsboro, OR, USA

Proceedings of the 17th International Conference on VLSI Design (VLSID'04)
1063-9667/04 $ 20.00 © 2004 IEEE

```
1   Procedure ACO
2      Initialize parameters and
3      pheromone trails
4      for colony := 1 to ncolonies do
5         for ant := 1 to m do
6            Generate tour
7            Compute cost
8            if cost < min_cost
9               Save solution
10              min_cost = cost
11           end if
12           Update pheromone trails
13        end for
14        Evaporate
15     end for
16  end Procedure
```

Figure 1. ACO algorithm

at time $t+1$:

$$\tau(r,s;t+1) = \rho\tau(r,s;t) + \Delta\tau(r,s;t) \tag{1}$$

where ρ is a parameter such that $1-\rho$ represent the evaporation coefficient. In order to avoid unlimited accumulation of pheromone, $\rho \in (0,1)$. The trail reinforcement is the sum of the reinforcements generated by all the ants in the colony:

$$\Delta\tau(r,s;t) = \sum_{k=1}^{m} \Delta\tau_k(r,s;t) \tag{2}$$

where m is the number of ants in the colony and $\Delta\tau_k(r,s;t)$ is the quantity of pheromone laid by the k-th ant at time t, defined as:

$$\Delta\tau_k(r,s;t) = \begin{cases} \dfrac{Q}{L_k}, & \text{if the } k\text{-th ant uses the edge } (r,s) \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

where Q is a constant and L_k is the tour length of the k-th ant.

Let $\eta(r,s)$ be the *visibility* between the vertex r and the vertex s. $\eta(r,s)$ is defined as $\eta(r,s) = 1/w(r,s)$, where $w(r,s)$ is the weight of the edge (r,s) value that does not change during the iterative process. The probability of an ant k to choose s as the next vertex when it is at the vertex r at time t is given by:

$$p_k(r,s;t) = \begin{cases} \dfrac{\tau^{\alpha}(r,s;t)\eta^{\beta}(r,s)}{\sum_{u\in M_k}\tau^{\alpha}(r,u;t)\eta^{\beta}(r,u)}, & \text{if } s \in M_k \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

where M_k is the set of vertices not visited yet by the ant k, α and β are two parameters that control the relative importance of trail versus visibility. The basic ACO algorithm is shown in Figure 1. $ncolonies$ is the number of colonies (iterations) and m is the number of ants in a colony.

3. ACO Algorithm for Overlap Removal

The proposed technique uses the *constraint graphs* (directed acyclic graphs) to derive an overlap free placement. The *vertical* constraint graph $G_V(V_V, E_V)$ sets the y-coordinates of the cells, while the *horizontal* constraint graph $G_H(V_H, E_H)$ sets the x-coordinates. A macrocell placement is overlap free if for all the cell pairs $\{i,j\}$, there is an edge (i,j) or (j,i) in one of the constraints graphs (horizontal or vertical constraint graph).

The overlap removal algorithm is based on the ACO algorithm presented in Figure 1.

3.1. Visibility Graphs

In this section, the concept of visibility is adapted to macrocell placement. The visibility between two cells i and j is defined as a heuristic value which allows to evaluate the relevance of edges in the redundant constraint graphs, determined by the relative positions of the cells. Since there are two constraint graphs, two visibility graphs (η_H and η_V) must be used. Two cells may have the following relative overlap:

1. *Horizontal overlap*: The projections of the cells on the horizontal axis overlap.
2. *Vertical overlap*: The projections of the cells on the vertical axis overlap.
3. *Physical overlap*: Both horizontal and vertical overlaps are present.
4. *No overlap*: There is no vertical or horizontal overlap between the cells.

Let us define the weight of an edge $\eta_H(i,j)$ in horizontal visibility as the amount of vertical overlap between cells i and j and the weight of an edge $\eta_V(i,j)$ in vertical visibility as the amount of horizontal overlap. The direction of an edge in the visibility graphs is determined by the relative positions of the cells: from the cell with the lowest center to the other cell for η_V and from the cell with the left most center to the other one for η_H. For example, if the center of cell i is below the center of cell j and there is a horizontal overlap between the cells, η_V contains the edge (i,j). All the weights are normalized by dividing them with a constant value D_{avg}, defined as the average cell dimension.

The visibility definition is extended by allowing the relative overlap between two cells to have negative values:

$$\begin{aligned} V_{ovlp}(i,j) &= (Dx_i + Dx_j)/2 - H_{dist}(i,j) \\ H_{ovlp}(i,j) &= (Dy_i + Dy_j)/2 - V_{dist}(i,j) \end{aligned} \tag{5}$$

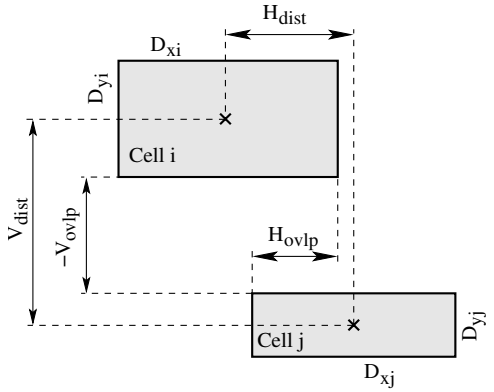


Figure 2. Visibility calculation

where Dx_i and Dy_i are the dimensions of cell i and $H_{dist}(i,j)$ and $V_{dist}(i,j)$ are the distances between the centers of the cells i and j on the two axes of coordinates. Note that if the cells do not physically overlap, the overlap value is negative. The weight of edges in the visibility graphs between the two cells i and j is determined by:

$$w(\eta_V(r,s)) = (V_{ovlp}(i,j) - Ovlp_{min} + \delta_{ovlp})/D_{avg}$$
$$w(\eta_H(r,s)) = (H_{ovlp}(i,j) - Ovlp_{min} + \delta_{ovlp})/D_{avg} \quad (6)$$

where (r,s) is (i,j) or (j,i), $Ovlp_{min}$ is the minimum between $V_{ovlp}(i,j)$ and $H_{ovlp}(i,j)$, and δ_{ovlp} is a constant value used to make both V_{ovlp} and H_{ovlp} have positive values. δ_{ovlp} can be the average cell size D_{avg}. The direction (r,s) of the edge is determined by the relative positions of the cells.

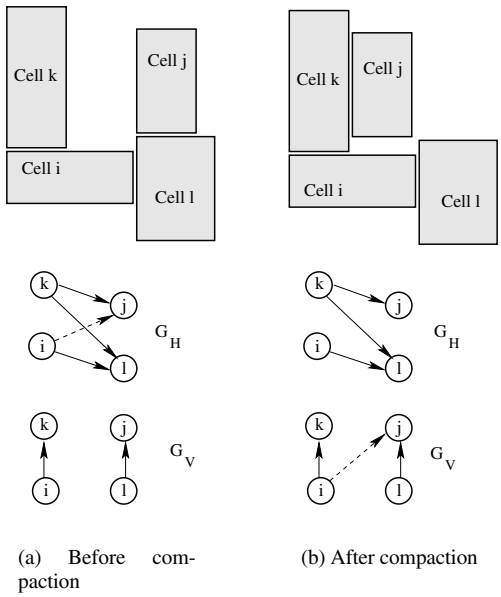
Figure 2 shows an example of visibility calculation for two cells i and j. In this case, the visibility graphs have two edges: (i,j) in η_H and (j,i) in η_V with the weights given by Equation 6. The edges (j,i) in η_H and (i,j) in η_V have the weight zero.

3.2. Construction of the Constraint Graphs

Let us define $\mathcal{M}(i,j)$ as the set of edges in the visibility graphs between the cells i and j. Since only one edge in both constraint graphs is enough to have an overlap free placement, the algorithm will choose one edge from $\mathcal{M}(i,j)$ to be added in the constraint graphs. The algorithm uses two trail arrays τ_H and τ_V to record how cost effective is the usage of each edge from $\mathcal{M}(i,j)$. Initially, all the fields in τ_H and τ_V are set to unity.

For all pairs of vertices $\{i,j\}$, if there are positive weighted edges in $\mathcal{M}(i,j)$, an ant k will choose one edge to be added to the constraint graphs with the probability:

$$p_k(r,s) = \frac{\tau^\alpha(r,s)\eta^\beta(r,s)}{\sum_{(u,v)\in\mathcal{M}(i,j)} \tau^\alpha(u,v)\eta^\beta(u,v)} \quad (7)$$



(a) Before compaction (b) After compaction

Figure 3. Example of placement compaction

where (r,s) is an edge from $\mathcal{M}(i,j)$. $\tau(r,s)$ and $\eta(r,s)$ are $\tau_V(r,s)$ and $\eta_V(r,s)$ or $\tau_H(r,s)$ and $\eta_H(r,s)$, depending on the graph where the edge (r,s) belongs. If all the edges in $\mathcal{M}(i,j)$ have the weight zero, no edge will be added. The cells with in-degree equal to zero in one of the constraint graphs will have the corresponding coordinate value set to zero and they will be the first cells processed when the cell positions are calculated. The cell positions can be easily computed applying a procedure similar to breadth-first search algorithm on the constraint graphs, starting with the vertices with in-degree zero. If x_i is the coordinate of a parent, all its children will have $x = x_i + Dx_i$, where Dx_i is the dimension of cell i.

3.3. Local Optimization of Cell Placement

Since an ant chooses the edges stochastically in the two constraint graphs, some suboptimal constraint edges can be selected. This will generate a placement with unused space as shown in Figure 3(a). Let us consider that the y coordinates of the cells are computed. The placement can be optimized horizontally by testing if the parent and the child can have vertical overlap, i.e., the vertical distance between the cells is greater than zero. When no vertical overlap is possible, the edge between the two cells in the horizontal constraint graph is removed and an edge between the two cells is added to the vertical constraint graph. The direction of the edge is from the lower cell to the upper cell. Similarly, the area can be optimized vertically when x coordinates are computed first. Figure 3 shows how the compaction is done. Cell j is the child of cell i in the horizon-

```
1  Procedure Place & Compact
2    Calculate the y coordinates
3    Calculate the x coordinates
4       and compact horizontally
5    Reset the y coordinates
6    Calculate the y coordinates
7       and compact vertically
8  end Procedure
```

Figure 4. Place and Compact procedure

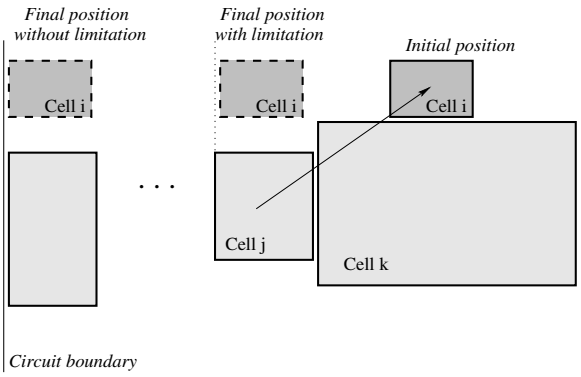


Figure 5. Limiting the cell sliding

tal constraint graph $G_H(E_H, V_H)$. Because the two cells do not have vertical overlap, the edge is removed from the horizontal constraint graph and an edge is added in the vertical constraint graph $G_V(E_V, V_V)$. The direction is from i to j because the center of i is below the center of j. When this operation is performed, the coordinate calculation must be performed three times as shown in Figure 4. The procedure `Place & Compact` processes first the vertical constraint graph, then the horizontal constraint graph, then again the horizontal constraint graph. In order to have a better compaction in both directions, the algorithm alternates the order in which the constraint graphs are processed for each iteration. Figure 4 shows one of the two possible forms of the procedure. A problem that may occur during this phase is the sliding of cells from the neighborhood of the initial position to the circuit boundary. For example, cell i shown in the Figure 5 will slide from the initial position because it has the biggest y value and all the edges in the horizontal constraint graph incident to i will be removed by the local optimization procedure. This problem can be solved by setting a minimum position for a cell when an edge is moved from a constraint graph to the other. Consider cell i in Figure 5. The horizontal constraint graph contains the edge (j, i) which will be removed and an edge will be added in the vertical constraint graph. When the edge (j, i) is removed, the minimum x value of the cell i is set as the x coordinate of the cell j. This can be extended to all the edges in the horizontal constraint graph incident to i:

$$X_{min}(i) = min\{x(k)|(k, i) \in \mathcal{R}_H(i)\} \tag{8}$$

where $\mathcal{R}_H(i)$ is the set of edges incident to i which were removed from the horizontal constraint graph. Similarly, it can be set a minimum value for the y coordinate.

3.4. Trail Update

The trails are updated using Equation 1. In order to avoid stagnation, the trails are not allowed to have values below a minimum bound ($\tau_H(r, s) \geq 0.1$ and $\tau_V(r, s) \geq 0.1$ in this work). The reinforcement trail intensity $\Delta\tau$ is determined by:

$$\Delta\tau(r, s) = \begin{cases} \dfrac{Q}{Cost_K}, & \text{if the ant uses the edge } (r, s) \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

where Q is a constant. $Cost_K$ is the cost of the best placement obtained by the colony K and is defined as:

$$Cost_K = w_1 \cdot A/A_0 + w_2 \cdot L/L_0 \tag{10}$$

where A is the total circuit area, A_0 is the initial circuit area, L is the total wire length, and L_0 is the initial total wire length. w_1 and w_2 are the weights of the area and wire length and can be specified by the user. We used $w_1 = 1$ and $w_2 = 2$ in order to have a better wire length result. Since the cost value is close to three in this case, the constant Q from Equation 9 can be set to unity. $\Delta\tau(r, s)$ is less than one, but does not have a very small value that will make the trail values go to zero when the evaporation coefficient is high. The cost of the current placement is compared with the best cost so far. If the current cost is lower, the current placement is saved and the best cost updated.

Figure 6 summarizes all the steps of the overlap removal procedure. $ncolonies$ is the number of colonies used and $nants$ is the number of ants in a colony.

4. Improving the Execution Time

The execution time of the search procedure, similar to breadth-first search algorithm, which calculates the positions of the cells can be reduced by using a special queue to store the vertices to be processed. The constraint graphs contain transitive edges which cause some vertices to be processed several times. The number of times a vertex is processed can be reduced if the queue is searched when a new vertex is added. If the vertex is already in the queue, it will be moved to the tail (the last one to be processed). The search can be done in constant time when the queue is implemented as an array whose elements are also linked by pointers.

If all the cell pairs are considered when the visibility graphs are constructed, then the time complexity for `Generate constraint graphs` and `Place &`

```
1    Procedure Overlap Removal
2      Initialize parameters and
3      pheromone trails
4      Generate the visibility graphs
5      for colony := 1 to ncolonies do
6         for ant := 1 to nants do
7            Choose cell orientation
8            Generate constraint graphs
9            Place & Compact
10           Compute cost
11           if cost < best_cost
12              Save placement
13              best_cost = cost
14           end if
15        end for
16        Update pheromone trails
17        Evaporate
18     end for
19   end Procedure
```

Figure 6. Ant based overlap removal algorithm

Compact functions (see Figure 6) is $O(n^2)$, where n is the number of cells. The number of ants in a colony is typically proportional to n. Since the number of colonies does not exceed a constant limit, the overall time complexity of the algorithm is $O(n^3)$. This limit can be improved by reducing the number of cell pairs that generate edges in the visibility and trail graphs. When the distance between two cells is big, the probability that these two cells will overlap during the placement is close to zero. Let us define a visibility range $MaxRange$. If the distance between two cells is longer than $MaxRange$, no edges will be added in the visibility graphs. When $MaxRange$ is equal to the chip size, all cell pairs will generate edges in the visibility graphs and the corresponding adjacency matrices are dense.

Let us consider that the visibility range is several times the average cell size:

$$MaxRange = k \cdot D_{avg} \tag{11}$$

where k is a constant. This implies that the cells that will generate edges in the visibility graphs with any given cell are included within a $2kD_{avg} \times 2kD_{avg}$ window. The average number of cells covered by this window is the window area divided by the average cell size $4k^2D_{avg}^2/D_{avg}^2 = 4k^2 = constant$. Therefore, the number of cell pairs that will generate an edge in the visibility graphs is reduced, in average, to $4k^2n = O(n)$. This observation shows that the adjacency matrices of the visibility graphs are sparse when only the useful edges (edges between close cells) are considered. The overall average time complexity in this case will be $O(n^2)$.

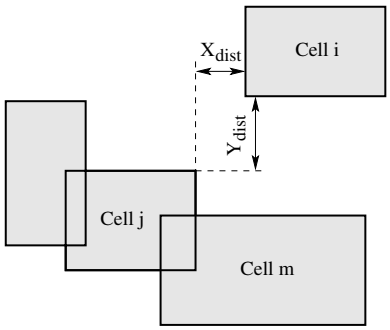


Figure 7. Determining the minimum visibility range

Circuit	Cells	Nets	Initial	
			Wire (mm)	Area (mm^2)
Compress	35	174	33.3	0.334
Find	58	268	89.2	0.698
Fifo	63	296	83.3	0.866
Elliptic	94	531	191.1	1.013
Shuffle	104	488	115.2	1.010
Ami33	33	119	8.0	0.027
Ami49	49	408	108.8	0.806
Playout	62	2,506	617.0	1.475

Table 1. Benchmark characteristics and initial values

Reducing the number of edges in the visibility graphs will reduce the execution time, but due to cell movement from the initial position and local improvement presented in Section 3.3 the resultant placement may have overlaps. We have used two methods to ensure that the final layout will not be overlap free. The first one which was presented in Section 3.3 limits the sliding of cells which are placed on top and right edges of the circuit.

The probability that a cell will overlap with another cell decreases with the increase of the $MaxRange$. When $MaxRange$ is equal to the circuit size, this probability is zero. However for most of the cells a much smaller limit is enough. We used an adaptive technique that finds the minimum required value for the visibility range. Consider the cell i and the situation presented in Figure 7. Let $\mathcal{C}(i)$ be the set of cells placed below and to the left of cell i. Let j be the cell in $\mathcal{C}(i)$ so that the distance $D(i,j) = Xdist(i,j) + Ydist(i,j)$ is minimum. The visibility range is determined by:

$$MaxRange(i) = \max\{Xdist(i,j), Ydist(i,j)\} + k \cdot D_{avg} \tag{12}$$

where k is a constant with a value between 2 and 5. This method will increase the visibility range for the isolated cells. If the cell overlaps with other cells, the visibility range will be $k \cdot D_{avg}$.

Circuit	Ant based (50 colonies/100 ants)			Ant based (100 colonies/200 ants)			Constr. reduction			SA based		
	Wire (mm)	Area (mm^2)	Run (m:s)	Wire (mm)	Area (mm^2)	Run time	Wire (mm)	Area (mm^2)	Run time	Wire (mm)	Area (mm^2)	Run time
Compress	31.7	0.274	12s	31.2	0.274	55s	34.6	0.306	2s	31.5	0.303	1m:43s
Find	86.4	0.585	23s	84.8	0.567	1m:34s	90.8	0.625	3s	85.6	0.600	4m:47s
Fifo	79.4	0.750	24s	76.1	0.697	1m:38s	83.4	0.784	3s	77.2	0.752	5m:29s
Elliptic	222.4	1.366	1m:17s	215.0	1.251	5m:21s	240.8	1.483	11s	220.5	1.422	17m:24s
Shuffle	120.0	1.025	1m:12s	116.3	0.988	4m:15s	126.8	1.231	24s	125.3	1.116	15m:21s
Ami33	7.0	0.019	13s	6.6	0.019	50s	8.4	0.024	2s	8.2	0.027	1m:29s
Ami49	121.4	0.771	30s	117.4	0.748	2m:01s	152.1	1.101	2s	112.8	0.819	4m:24s
Playout	842.1	1.909	1m:29s	801.3	1.883	5m:14s	970.2	2.260	28s	771.5	1.998	18m:07s

Table 2. Experimental results

5. Experimental Results

We tested the proposed overlap removal procedure on a set of layouts generated by a force-directed macrocell placement algorithm. The circuits used as benchmarks are: (1) synthesized by a behavioral synthesis system from high-level specification; (2) MCNC benchmarks. Table 1 shows the benchmark characteristics: number of cells, number of nets, and the initial value for area and wire length. The results were obtained on a SUN ULTRASPARC 30 Workstation with 300MHz processor and 128MB RAM. The program is written in C++.

We considered that both the visibility and trails to have equal importance, the α and β coefficients from Equation 7 being set to unity and the evaporation coefficient to $1 - \rho = 0.1$. The results of the ant based overlap removal procedure were compared with the original layouts and those generated by the constraint reduction algorithm proposed by Asato and Ali [7].

Table 2 compares the area and the estimated wire length of the proposed approach for 50 colonies and 100 ants and 100 colonies and 200 ants with the placement generated by the constraint reduction algorithm and the placement generated by a low temperature simulated annealing based approach. The initial solution for the simulated annealing based approach is obtained by taking, for any pair of cells, the edge with the largest weight from the visibility graphs. The moves consists on replacing one edge from a constraint graph with the corresponding edge in the other constraint graph. The wire length is estimated in all cases by using the half-perimeter method. The results show that the ant based algorithm outperforms the constraint reduction algorithm which is stuck in local minima. Also, the constraint reduction algorithm does not consider the wire length during the overlap removal process which affects its performance. One can observe that the increase of the area and wirelength for the proposed method is small. In some cases the values are smaller than those of the initial placement. This can be explained by a better packing and local optimization of the cell

6. Conclusions

In this paper, a new macrocell overlap removal procedure has been proposed. The novelty of the approach lies in the use of the ant colony optimization heuristic to search the solution space in the neighborhood of an initial placement. The test results show that our method performs better than the existent techniques. In some cases the placement is improved over the initial placement.

References

[1] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. "VLSI module placement based on rectangle-packing by the sequence-pair". *IEEE Trans. Computer-Aided Design*, pages 1518–1524, Dec. 1996.

[2] P-N. Guo, T. Takahashi, C-K. Cheng, and T. Yoshimura. "Floorplanning using a tree representation". *IEEE Trans. Computer-Aided Design*, pages 281–289, Feb. 2001.

[3] Y-C. Chang, Y-W. Chang, G-M. Wu, and S-W. Wu. "B*-Trees: a new representation for non-slicing fborplans". In *Proc. Design Automation Conf.*, pages 458–463, 2000.

[4] G. Sigl, K. Doll, and F.M. Johannes. "Analytical placement: a linear or a quadratic objective function?". In *Proc. Design Automation Conf.*, pages 427–432, 1991.

[5] F. Mo, A. Tabbara, and R. K. Brayton. "A force-directed macro-cell placer". In *Proc. Int. Conf. Computer-Aided Design*, pages 177–180, 2000.

[6] J. Vijayan and R-S. Tsay. "A new method for fbor planning using topological constraint reduction". *IEEE Trans. Computer-Aided Design*, pages 1494–1501, Dec. 1991.

[7] B. Asato and H.H. Ali. "An improved fbor planning algorithm using topological constraint reduction". In *Proc. 38th Midwest Symposium on Circ. and Systems*, pages 310–313, 1996.

[8] S. Nag and K. Chaudhary. "Post-placement residual-overlap removal with minimal movement". In *Proc. DATE Conference*, pages 581–586, 1999.

[9] Dorigo M., V. Maniezzo, and A. Colorni. "The ant system: optimization by a colony of cooperating agents". *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, pages 29–41, 1996.