Olympiads in Informatics, 2015, Vol. 9, 193–203
DOI: http://dx.doi.org/10.15388/ioi.2015.15

Ant Colony Optimisation Applied to Non-Slicing Floorplanning

Mirzakhmet SYZDYKOV[1], Madi UZBEKOV[2]

[1] *Kazakh National Technical University named after K.I. Satpayev*
Satpayev Str. 22a, Almaty, Kazakhstan 050013
[2] *Kazakh Economical University named after T.Ryskulov*
Zhandosov Str. 55, Almaty, Kazakhstan 050035
e-mail: rbtinf@gmail.com, *uzbekm7@gmail.com*

Abstract. In this article experimental results are provided for a very-large-scale integration (VLSI) floorplan design problem. Given is a set of modules to be placed non-overlapping on a 2-dimensional rectangular plane. We use ant system simulation as a heuristics to produce feasible layouts in order to minimize the total unused area. The algorithm differs from many others in that fact that it produces non-slicing floorplan. Our experimental results show comparable results of previous methods using ant colony optimization (ACO) in VLSI design. For this purpose we define the "interior" structure for a geometrical computation of module positions.

Keywords: algorithm, ant system, interior, optimization, VLSI, floorplanning.

1. Introduction

At the present time there are several known methods to solve the floor planning problem in VLSI circuit design using ACO heuristics (the main algorithm's described in Dorigo *et al.*, 1996):

1) With a temperature-aware constraint (Luo and Sun, 2007).
2) With a clustering constraint (Chiang, 2009).
3) With a non-overlapping constraint (Alupoaei and Katkoori, 2004).

The last method also provides a solution that removes overlaps of placed modules and reduces the total area and wire length. Our method uses a similarly incremental approach to build constraint graphs and place modules in vertical (to the bottom) or horizontal direction (to the right). This method utilizes the interior structure in order to find a relative placement of the module. In our problem the placement has a single constraint: modules do not overlap each other. We will use a mathematical notation to represent a target function in order to minimize the total unoccupied empty space, which is further defined as a *dead space*.

The optimization methods using ACO heuristics are widely discussed in recent publications:

For circuit partitioning in VLSI design (Arora and Lall, 2013).

For routing optimization with *tabu* search(Yoshikawa and Otani, 2010).

Our method mainly differs in the definition of *visibility* and *distance* functions used in original algorithm (it is better described in Section 3) from the algorithms above which in fact are driven models with modified *core* functions.

1.1. *Problem Definition*

The floor planning problem consists of a set of modules on an integral circuit to be arranged on a planar area in such a way that they will not overlap each other while the occupied areas' measurements, which are given by their formulas, are to be optimized. We solve the problem where the total space unoccupied by the modules is minimized with a non-overlapping constraint by an experimental algorithm. The minimization function is given as a ratio. This can be better defined with an equation:

$$\frac{\sum_{m \in M} Area(m)}{Area(R)} \rightarrow min \tag{1}$$

where M is a set of placed modules and R is a rectangle bounding the placement. The function $Area\ (m)$ is the total area occupied by the module m, whereas the $Area\ (R)$ is the total area of the rectangular board. In our problem the module is given by its bounding box, here it does not actually matter what is the physical shape of the element. We also consider the total area to be the area of the bounding box containing all the placed modules. The minimization of the function (1) is achieved by minimizing the total dead space, which in experimental purposes is measured as a percentage ratio of the part of the bounding box containing all the modules. In can be better represented as:

$$Area\ (R) : R = Bounding\ Box\ (\ Union\ \{each\ m\ in\ M\})\ ), \tag{2}$$

$$Target\ function\ (1) \rightarrow min, iff\ "Dead\ space"\ (\%) \rightarrow min, \tag{3}$$

1.2. *Known Solutions*

There are number of methods to generate a feasible placement for the given set of modules. Most of the methods use specific structures like a B-Tree (Sivaranjani and Kawya, 2013), polish notation or Corner Intersection Sequence (CIS) (Hoo *et al.*, 2013) to internally represent a valid placement. These structures can represent a *slicing floorplan* where the rectangular area of placement can be recursively divided into two parts by a horizontal or vertical line, while each of the modules is within the bounds of the final rectangles produced by an algorithm (Fig. 1).

Our method differs from listed above in fact that it produces non-slicing floorplan. The main difference is also in type of structures used in algorithm. They will be discussed in the next section.

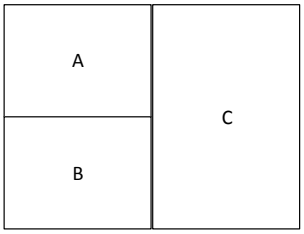

Fig. 1. An example of a slicing floorplan.

2. Data Structures Used in Algorithm

The algorithm uses two types of data structures in order to solve the optimization task – interior and constraint graph. These structures are of planar geometric (interior) and abstract (constraint graph) type. We present simple algorithms to construct them. This generally does not limit the variety of type of data structures which could replace interior and constraint graph for their main purpose to place the module and minimize the value of function (1).

The purpose of interior structure is to store the modules' placement. The non-overlapping condition is to hold true. Using this structure we have to answer queries to find the coordinates of the side projections on vertical or horizontal axis.

The constraint graph structure is to represent the abstract order of module placement relative to the horizontal or vertical axis. The graph is to be acyclic. Using this structure we put all the modules in the placement in abstract topological order. This is necessary to pack the modules after the new module is placed. This structure is to answer the queries to find the placement coordinates of the leftmost bottom corner (x- and y-coordinate for side projections) of the module as if they would be packed without overlapping each other by a physical power vector coming from outermost space on a plane (i.e. the most upper right area).

On the Fig. 2, the packing scheme is presented, the vectors are denoted as P(X) and P(Y), for vertical and horizontal direction respectively.

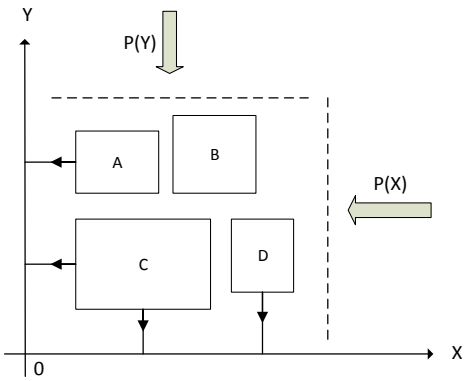

Fig. 2. The packing scheme.

2.1. *Constraint Graph*

A constraint graph is the structure we use to represent the floorplan as an acyclic directed graph. The constraint graph represent the order of modules' placement relatively to the horizontal or vertical axis. Thus the constraint graph can be either horizontal or vertical. This can be better illustrated if we would draw these graphs for a module placement on Fig. 1 (Fig. 3).

Here the "0"-mark stands for the artificial starting element which in fact is a parental node having no incoming edges. Physically on a plane this means that the leftmost or the most bottom modules are to be connected to this parental node as it can be seen on the example diagrams (Fig. 3).

2.2. *Interior Structure*

The interior of the current feasible placement may be described as a set of vertical or horizontal ranges representing the projections of modules taking into account their relative order. To better understand the structure of interior study the example in Fig. 4.

This structure can be effectively used to build constraint graphs or to detect the relative position of the placing module.

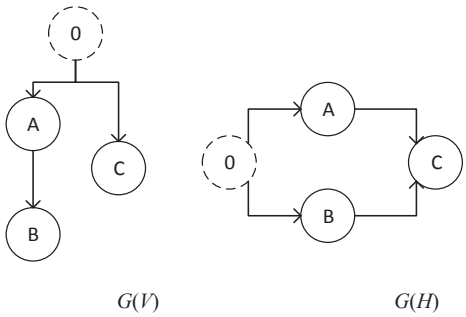

Fig. 3. An example horizontal and vertical constraint graphs.

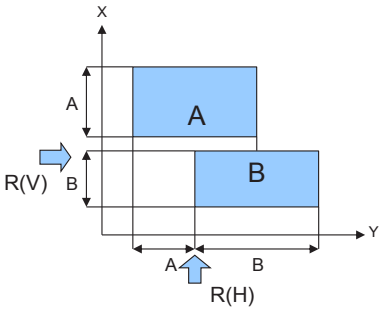

Fig. 4. Example of placement modules and their vertical R(V) and horizontal R(H) interiors.

The interior structure is an ordered list of ranges which may also be used in constraint graph detection. This is mainly because of that fact that each element of this list is a segment on the *X*- or *Y*-plane with *Z*-axis as the other (minor) coordinate. This can best be seen in Fig. 4 in the red line. To build the graph we have to detect if the interior's "red line" intersects the next item which is to the left or at the top according to parameter *Z*. If yes, then there would be a relation between the modules represented by segments in a constraint graph. This relation has direction according to the *Z*-axis. The axis may be either vertical or horizontal according to the constraint graph type. These types split the process of extracting the *x*- and *y*-coordinate for the module.

Mathematical Description of Interior Structure

The interior structure was designed to find a position of a module $m_i \in M / M_{Final}$ to be added to the right or bottom without overlapping an arbitrary element which is already included in the final placement M_{Final}. More precisely, the interior can be viewed as an outermost horizontal or vertical line lying on the edges of modules in placement, viewed from right or bottom side on a plane. Because the algorithm iteratively produces the feasible placement, the interior structure needs to be updated. The interior I can be represented as a set of segments given by a vector of four values:

$$I = \{(L_i, R_i, Z_i, m_i) : I = 1..n, m_i \in M\}, \tag{4}$$

where L_i, R_i are left and right coordinates of the segment on a linear vertical or horizontal axis (this depends on the type of interior which can be either horizontal or vertical),

Z_i is a distance between the segment and parallel axis,

m_i is a module which covers the segment by its right or bottom edge.

Fig. 5, as is, gives an example of this structure on a geometric plane.

Below is an algorithm to update the interior structure according to the new module to be placed.

Please note, Z_i is a pre-determined value which does not change. For the *X*-interior it is obviously a *Y*-value of the bottom corner of the module and vice versa for the *Y*-interior.

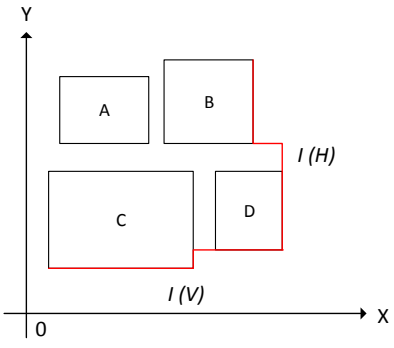

Fig. 5. View of an interior on a geometric plane.

Algorithm 1. Update interior structure.

Initial values: $I = \{\}$.
Input: The new module m_{New}.
Output: Interior I.
for each segment $a \in I$:
  if projection of a intersects edge of m_{New}:
    b = intersection result of a and m_{New};
    if $a.L < b.L$ then $I = I \cup \{(a.L, b.L, a.Z, a.m)\}$;
    if $b.R < a.R$ then $I = I \cup \{(b.R, a.R, a.Z, a.m)\}$;
    $I = I/\{a\}$;
  end if
end for

3. Ant System

In (Dorigo *et al.*, 1996) there is a proposed solution for a *Traveller-Salesman Problem* (TSP) problem using ant agents' simulation. This algorithm uses the measurement functions in order to get the probabilities of transitions between towns given on a planar area:

$tau_{i,j}(t)$ is an *intensity of trail* on edge (i, j) at time t.

The trail intensity is to be updated according to the following formula:

$$tau_{i,j}(t+n) = rho * tau_{i,j}(t) + \Delta tau_{i,j}, \quad (5)$$

where *rho* is a coefficient such that (1 - *rho*) represents the *evaporation* of trail between time t and $t + n$.

$$\Delta tau_{i,j} = SUM \{\Delta tau^{k}_{i,j} \mid k = 1..[Ants]\}, \quad (6)$$

where $\Delta tau^{k}_{i,j}$ is the quantity per unit of length of trail substance (pheromone in real ants) laid on edge (i, j) by the k-th ant between time t and $t + n$; these values is non-zero if ant uses edge (i, j) on his tour and equals value:

$$\Delta tau^{k}_{i,j} = Q / L_k, \quad (7)$$

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

The visibility $eta_{i,j}$ is defined as a quantity $1 / d_{i,j}$, where $d_{i,j}$ is a distance between towns i and j.

The transition probability from town i to town j for the k-th ant is defined as:

$$p^{k}_{i,j}(t) = (tau_{i,j}(t) * eta_{i,j}) / SUM \{ tau_{i,k}(t) * eta_{i,k} \mid k \text{ is allowed to be used in a tour}\}, \quad (8)$$

Please note: the functions (5)–(8) are *core* functions required to create a base for simulation model which uses results of computation to make a decision.

Our method differs only in the definition of *visibility* and *distance* functions:

$$Viz\ (a, b) = \text{“Total Module Area”} / \text{“Total Area”}; \tag{9}$$

$$Distance\ (a, b) = Beta / Viz\ (a, b), \tag{10}$$

where the total module area is a cumulative sum of corresponding modules and total area represents the rectangular placement bounding box. The visibility function *Viz* (*a*, *b*) between modules *a* and *b* is a “visibility” used in the TSP algorithm. The distance function *Distance* (*a*, *b*) is a measure of divergence between modules *a* and *b*. This function is also used in this algorithm. *Beta* is defined as “Q” in Dorigo *et al.*, 1996 (equation (3)), in this paper it is equation (7). It is used as a constant of any positive value. In our algorithm it always equals one.

4. Basic Algorithm

The matrix of trained ants’ probability values to search the best placement is represented as a product of dimensions $2N \times 2N \times P$, where P is a set of values – {BOTTOM, RIGHT}. We use this notation in order to include the possible flipped (sides of a module rotated 90 degrees) orientation of the corresponding module. In this case the module index is multiplied by two. The set P represents the possible relative placement of modules. Thus, the pheromone matrix (Dorigo *et al.*, 1996) in our algorithm is a multi-dimensional array. In order to take into account that fact that the new module can be placed to the right or to the bottom, the dimension degree is increased by using a set P accordingly. The dimension of the rectangular matrix is also increased (in fact it is multiplied by two) with respect to that fact that the modules can be rotated. For the indexes 1..2N the following is assumed:

(1) Indexes in form of $2k + 1$ ($2k + 1 <= 2N$) are original modules.
Example: 1, 3, 5, ...
(2) Indexes in form of $2k$ ($2k <= 2N$) are rotated modules.
Example: 2, 4, 6, ...

The algorithm is similar to the ant colony best path search simulation described in (Dorigo *et al.*, 1996). On every step the new module can be placed either to the right or bottom relatively to any module from the set built using ant simulation. The new module is to be placed according to the minimal value of the distance function. I.e., from all the distance values we choose the module corresponding to the minimal value. To solve the problem of placement on the plane the interior structure is used which on every step determines the position of the new module. This can be done in $log\ (N)$ number of operations using a binary search.

When the placement is created, an additional operation is applied. We call it *packing* and it uses the constraint graphs of the placement to rebuild it according to the topological structure of the graph. More precisely, the constraint graph is used to rebuild the placement in order to pack it. This is achieved by computing the values of *X*- and *Y*-coordinates of the modules according to the graph structure (horizontal or vertical). This can be done using a *Breadth-first search* (BFS). Thus the possible residual dead space is excluded from area occupied by the newly placed modules.

The complexity of the solution is $O\ (NC\ N_{Ant}\ N^3\ log\ (N))$, where NC is the number of outer iterations, N_{Ant} – number of artificial ants and N – the number of modules in final placement.

5. Experimental Results

In this section we give the graphical plot of the obtained results using the described method of ant colony optimization of modules to be arranged with no overlaps. The benchmark tests included test cases from *CompaSS* software package (CompaSS, 2004–2005). Below are graphical plots of the algorithm results for the cases *AMI33* and *AMI49* presented on Fig. 6 and Fig. 7 respectively.

The practical observations show that algorithm gives better results if the number of artificial ants and number of outer iterations is increased. This can be better analysed from the results presented in Table 1.

On the diagram below (Fig. 8) the results are visualized for each iteration (one line for each value). The ants count values are on horizontal axis by 5 ants per unit and dead space percentage values are on vertical axis.

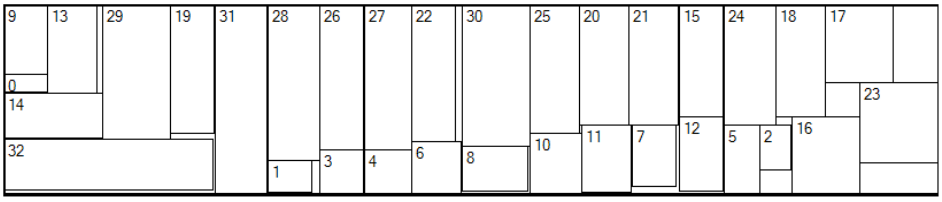

Fig. 6. Physical placement of AMI33, Unused area = 6.888%.

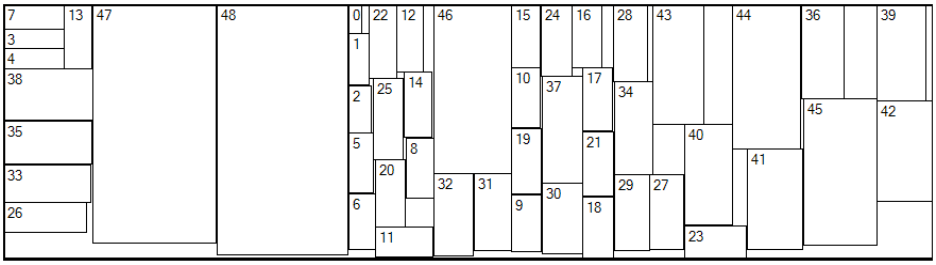

Fig. 7. Physical placement of AMI49, Unused area = 10.621%.

Table 1
Experimental results for AMI33 with varying parameters

Module	Number of iterations	Number of ants	Dead Space (%)	Running time
AMI33	5	5	32,24%	890 ms
		10	12,20%	1 sec. 389 ms
		15	6,89%	1 sec. 988 ms
		20	17,25%	2 sec. 645 ms
		25	6,89%	3 sec. 145 ms
	10	5	28,72%	1 sec. 795 ms
		10	12,20%	2 sec. 948 ms
		15	6,89%	4 sec. 260 ms
		20	6,89%	5 sec. 184 ms
		25	6,89%	6 sec. 392 ms
	15	5	21,00%	2 sec. 473 ms
		10	17,25%	4 sec. 643 ms
		15	6,89%	6 sec. 303 ms
		20		7 sec.
		25	6,89%	9 sec. 811 ms
	20	5	12,20%	3 sec. 567 ms
		10	6,89%	6 sec. 21 ms
		15	12,20%	8 sec. 177 ms
		20	6,89%	10 sec. 831 ms
		25	6,89%	12 sec. 513 ms
	25	5	19,86%	4 sec. 407 ms
		10	13,66%	7 sec. 447 ms
		15	12,20%	9 sec. 682 ms
		20	6,89%	13 sec. 118 ms
		25	6,89%	16 sec. 321 ms

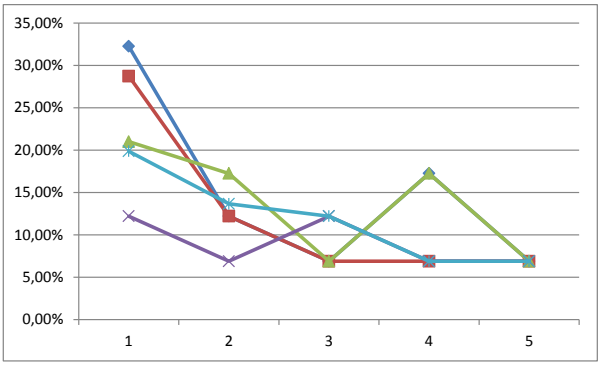

Fig. 8. The visualization of results in Table 1

5.1. *Generalizing Algorithm*

By the generalization of the described algorithm we mean the application of heuristics plan for a large data set (estimated as more than 10K nodes). These data sets were proposed for the contest held at the *International Symposium on Physical Design* (ISPD, 2005). Practically, the algorithm can be maintained effectively even for large data sets if we apply the clustering paradigm. This paradigm includes the following steps to be applied:

1. Select a set of clusters to divide the entire list of modules independently:

$$C\text{: } Union\ \{each\ c\ in\ C\} = C\ \&\ Intersection\ (each\ a\ in\ C, each\ b\ in\ C \mid a\ !=\ b) = \{\}; \tag{11}$$

2. Apply locally the ACO algorithm for each cluster using the set of modules in cluster as an input data:

$$Local\ Placement = Union\ \{ACO\ (each\ c\ in\ C)\}; \tag{12}$$

3. For the list of placements obtained from step 2 create a list of bounding boxes:

$$Global\ List = Union\ \{\text{“}Bounding\ Box\text{”}\ (each\ p\ in\ \text{“}Local\ Placement\text{”})\}; \tag{13}$$

4. Apply the ACO algorithm globally:

$$Global\ Placement = Union\ \{ACO\ (each\ p\ in\ \text{“}Global\ List\text{”})\}; \tag{14}$$

These steps can be applied recursively to large data sets, if we would use the algorithm for the clusters as the input data, which in turn may be a result of ACO algorithm for the other clusters. These clusters at their finite hierarchy are modules representing the input data for the global algorithm.

5.2. *Conclusion and Further Work*

The working algorithm produces better results when the number of ants is increased. The further work includes the study of the application of clustering method to handle large amounts of data. This is not limited to the experiments when the list of constraints is extended as well as the list of semantic rules, for which the placement satisfies (for example, the final placement rectangle’s size and shape constraint).

Acknowledgements

We are glad to mention the editor of this article – prof. V. Dagienė (Vilnius University Institute of Mathematics and Informatics, Lithuania). We are grateful for the review due to which the article describing a novel algorithm became more readable and understandable including all the necessary and important information.

References

Alupoaei, S., Katkoori, S. (2004). Ant colony system application to macrocell overlap removal. *IEEE Transactions on VLSI Systems*, 12.

Arora, M., Lall, G.C. (2013). Circuit partitioning in VLSI design: an ant colony optimization approach. *International Journal of Advances in Engineering & Technology*, 6(1), 536–541.

Chiang C.-W. (2009). Ant colony optimization for VLSI floorplanning with clustering constraints. *Journal of the Chinese Institute of Industrial Engineers*, 26(6), 440–448.

CompaSS: *Compacting Soft and Slicing Packings* (2004–2005). `http://vlsicad.eecs.umich.edu/BK/CompaSS/`

Dorigo, M., Maniezzo, V., Colorni, A. (1996). The ant system: optimization by a colony of cooperating angents. *IEEE Transactions on Systems*, *Man*, *and Cybernetics*, *Part B*, 7–8.

Hoo, C.-S., Jeevan, K., Ganapathy, V., Ramiah, H. (2013). Ant system-corner insertion sequence: an efficient VLSI hard module placer. *Advances in Electrical and Computer Engineering*, 13(1).

ISPD: *The International Symposium on Physical Design* (2005). `http://archive.sigda.org/ispd2005/contest.htm`

Luo, R., Sun, P. (2007). A novel ant colony optimization based temperature-aware floorplanning algorithm. In: *Proceedings. Third International Conference on Natural Computation, ICNC* 2007. IEEE, 4, 751–755.

Sivaranjani, P., Kawya, K.K. (2013). Performance analysis of VLSI floor planning using evolutionary algorithm. *International Journal of Computer Applications*, *International Conference on Innovations in Intelligent Instrumentation*, *Optimization and Signal Processing* "*ICIIIOSP-2013*", 9, 42–46. `http://research.ijcaonline.org/iciiioes/number9/iciiioes1662.pdf`

Yoshikawa, M., Otani, K. (2010). Ant colony optimization routing algorithm with tabu search. In*: Proceedings of the International MultiConference of Engineers and Computer Scientists, IMECS 2010, March 17–19, 2010, Hong Kong.* III. 2104–2107.

M. Syzdykov currently works as a consultant in a small firm, graduated from Kazakh National Technical University named after K.I. Satpayev receiving a degree of engineer in systemotechnics. He's a participant of ACM ICPC 2004–2006 (NEERC subregion) and a CBOSS programming contest. He's also a certified specialist in such technologies like Oracle and Informatica Power Center. He has a working experience with data staging process optimization.

M. Uzbekov – currently works as a specialist in IT industry, Kazakhstan. Graduated from Kazakh Economical University named after T. Ryskulov. He's a certified specialist in such technologies like SAP and Informatica Power Center. He has a working experience with enterprise database and ETL systems like TeraData. He's also interested in programming challenges