Algorithms for Packing Soft Blocks of VLSI Systems

Submitted: 1st August 2012; accepted: 30th January 2013

Marcin Iwaniec, Adam Janiak

Abstract:
This paper contains a review of literature concerning the packing of hard-blocks (of fixed dimensions) and soft-blocks (of fixed area – changeable within specified constraints). These considerations are applicable to the designing of large scale integration chips. In order to solve the problem of packing soft-blocks, three algorithms are introduced and compared: simulated annealing, heuristic constructional algorithm based on five operations to improve packing quality and the algorithm which combines two previous algorithms. Experiments were conducted to compare these algorithms to the best from the literature.

Keywords: *soft-blocks, hard-blocks, VLSI, packing problem.*

1. Introduction

The continuous development of technology facilitates the lives of the users of modern devices, on the one hand, but also generates new problems to be faced by designers and engineers. One of such problems that challenge designers can be a physical synthesis of very large scale integration (VLSI) layouts. When one deals with billions of elements to be packed on a chip board (e.g. the most recent graphic card AMD Radeon HD 7970 has GPU with 4.31 billion transistors), the synthesis, which is the implementation of the physical packing of chip elements and interconnections according to a logical scheme, becomes a critical point at the stage of design because it directly influences the chip price. For example, a 1% increase in the size of an Intel Pentium 4 chipset would result in the annual increase of production costs amounting to $63.5 million (1.5%), whereas a size increase by 15% would induce the additional annual cost of $961 million (22%), see [7]. This implies the need to create algorithms which minimize the area needed to pack elements (or blocks, in general) on the chip surface at preset constraints in order to obtain the correct packing of elements and interconnections.

Two types of blocks have been distinguished: rectangle of fixed dimensions referred to as *hard-block* and rectangle of fixed area referred to as *soft-block* (additional constraints may also exist). Recent methods concerning block packing are largely based on so-called *sequence-pair*, introduced by [14], and a *constraint graph*. However, existing approaches for *hard-blocks* based on *sequence-pair* and *constraint graph* cannot perform for *soft-blocks*, especially if no starting dimensions are specified at the beginning. Nevertheless, it is necessary to consider algorithms solving the problem of minimizing the area of floorplans for *soft-blocks*, because they are frequently used at the early stages of chip designing, when not all the details have been provided.

Many algorithms for the *soft-blocks* packing problem may be found in the literature. Most of them were initially created to deal with the *hard-blocks* packing problem, and were adapted to *soft-blocks* as a consequence. Kirkpatrick in [11] described a simulated annealing (SA) algorithm which, since the first time it was used [15], has become one of the most frequently applied algorithms for packing *hard-blocks* ([19], [3], [21], [4], [9], [5]). Also other metaheuristics algorithms exist, e.g. genetic algorithm [16], evolutionary algorithm ([18], [12]) or an algorithm, which is based on the primary principles of ant colony optimization [6]. There are also some implementations of SA for *soft-blocks*. The simplest one randomly modifies block dimensions within possible range during the neighbourhood modification [13]. Kang and Dai in [10] presented another heuristic method of how to calculate optimum width to height ratio regardless of packing generated by simulated annealing algorithm. A more sophisticated method of blocks adjustment during neighbourhood modifications was designed in [8]. Block dimensions adjustment based on Lagrange relaxation algorithms was introduced in [23]. The branch and bound algorithm in [1] gives the best results – in some cases the solution derived from this algorithm is optimal, but the time needed for finding a solution is unacceptable.

In this paper we propose a quick heuristic (constructional) algorithm for floorplan minimization for *soft-blocks*. This algorithm requires an initial packing, which is then modified to eliminate empty spaces between blocks. The idea is based on the definition of *soft-blocks* – ability to change height and width within specified constraints. In this algorithm, were introduced some operations to improve the quality of the solution. The method was implemented in such a way, so that it could start from initial packing provided by any algorithm. In this paper, to generate the initial packing, simple implementation of simulated annealing algorithms based on the *sequence-pair* packing representation was used.

This paper is organized as follows. The problem is formulated in section 2. Then, section 3 provides a description of *sequence-pair* and *constraint graph* used to represent packing. Simulated annealing algorithm was described in section 4 and the essential

constructional algorithm in section 5. Section 6 demonstrates the results of experiments and it is followed by a summary and conclusion in section 7.

2. Problem Formulation

A set B = *{B_1, B_2, ..., B_i, ..., B_n}* of *soft-blocks* is given to be packed into rectangular area. Each block B_i is described by its area A_i as follows:

$$A_i := h_i * w_i \ , \qquad (1)$$

where h_i – block height, w_i – block width, furthermore, the ratio $k_i = \frac{h_i}{w_i}$ is in the range:

$$k_{imin} \leq k_i \leq k_{imax} , \quad i = 1, 2, \dots, n, \qquad (2)$$

where k_{min} and k_{max} are explicitly given parameters.

The packing which satisfies requirements (1) and (2), and no two blocks overlap, would be called a feasible one.

The problem is to find a feasible packing where dead-space (DS) is as close to 0% as possible. The minimized criterion DS is given as follows [17]:

$$DS = \frac{U - \sum_{i=1}^{n} A_i}{\sum_{i=1}^{n} A_i} \times 100\ [\%] \text{ , where} \qquad (3)$$

- U is a total area of a floorplan for packing and
- $\sum_{i=1}^{n} A_i$ is a sum of areas of packed blocks (A_i is an area of block *i*).

To solve this problem, the simulated annealing algorithm is used (section 4) modified for a considered problem and a constructional algorithm using the following operations: sliding, scaling, scaling with displacement, eliminating dead spaces and rotating layout (section 5). The third algorithm proposed in this paper combines both previous algorithms.

3. Floorplan Representation – *Sequence-pair* and *Constraint Graphs*

Both *sequence-pair* and *constraint graphs* are used to describe topological relation of blocks within a packing. A *sequence-pair* is an ordered pair of *n*-element block index sequences, both sequences contain exactly the same elements (indices of all blocks from the set *B)* arranged in specific order. The position of two particular elements in both sequences determines the topological relation of them. For given blocks indices *i,j ∈{1, 2, 3, ..., n}* and $i \neq j$, relation of blocks *i* and *j* will be horizontal, if in both sequences *i* comes first before *j* (4). Otherwise, the relation is vertical (5).

$$(< .., i, .., j, .. >,< .., i, .., j, .. >) \Rightarrow i \overset{H}{\rightarrow} j \qquad (4)$$
$$(< .., j, .., i, .. >,< .., i, .., j, .. >) \Rightarrow i \overset{V}{\rightarrow} j \qquad (5)$$

The topological relations of blocks can also be calculated by using constraint graphs, provided that one knows the height and width of every block. The horizontal (vertical) graph marked as *Gh* (Gv) has *n* vertices and *n* edges of specified weight. Vertices represent blocks, whereas edges represent topological relations. The edge between the vertex *i* and vertex *j* of weight equal to w_i (width of block *i*) is added to the horizontal graph when the condition (4) is fulfilled. If the condition (5) is fulfilled, the edge between vertex i and *j* of weight equal to h_i (height of block *i*) is added to the vertical graph. Thus one adds an edge to *Gh* and *Gv* for every pair of blocks in both sequences. Created constraint graphs are used to calculate a width (from the graph *Gh*) and a height (from the graph *Gv*) of the whole floorplan represented by these graphs. For this calculation one can use, for instance, the algorithm finding longest path in the graph [20].

In the Cartesian coordinate system the horizontal relation of *i* and *j* means that on the plane, block *j* is situated to the right of the block *i,* so $x_j \geq x_i + w_i$, where x_i is a horizontal coordinate of the lower left vertex of the block *i* and w_i is a width of the block *i*. In the same way, for the vertical relation, block *j* is situated over the block *i*, so $y_j \geq y_i + h_i$, where y_i is a vertical coordinate of the lower left vertex of the block *i* and h_i is a height of the block *i*. As a consequence, horizontal relation determines packing of blocks from left to right, and the vertical one from the bottom to the top.

4. Simulated Annealing Algorithm

The method described in [11] (or rather methodology, to be more specific) called simulated annealing, perfectly simulates the process, which is well-known in metallurgy as annealing (heating up an element and then cooling it down very slowly). Parameters required to apply this method are initial temperature (Tp), terminating temperature (ϵ), temperature change (α), neighbourhood function (to generate a new state) and goal function (calculation of state energy).

A great advantage of simulated annealing is that in the course of calculations, a worse solution can be accepted that can lead to a better one finally This method is partially based on a modified version of the Monte Carlo method called random walk method. This method assumes an acceptance of the obtained result that is worse than the best one, which had been found so far. The probability of acceptance of a worse solution is expressed by formula exp ($\frac{\Delta energy}{t}$), where Δenergy is a difference between the previous and current state energy, and t is a current temperature. Temperature dependence induces a higher probability that worse results shall be accepted at the first stage of method, when the temperature is higher. The lower the temperature is, the more often better results are accepted. As a consequence, during the final stage when the temperature is the lowest, the current solution can be improved as much as possible without replacement.

In this paper an algorithm was constructed, which is based on this method. During the annealing process, a sequence-pair is modified very simply, by the random exchange of two elements and the next obtained solution is evaluated by calculating the packing area (this is the cost function). Initial sequence-pair is generated randomly and used both in SA and the proposed algorithm described in the next section.

5. Constructional Algorithm

In this section, a new heuristic constructional algorithm is presented, which uses the operations described below to improve a solution. The algorithm starts from the same initial solution as simulated annealing and executes the following steps to improve this solution:

Step 1. Save the initial solution IS as a current solution CS (CS = IS) and as the best solution BS (BS = IS). Set rotation to 0° (R = 0°).

Step 2. Modify CS by executing operations in the following order: blocks sliding, blocks scaling, scaling with displacement, elimination of dead empty spaces; until the area is reduced as much as possible.

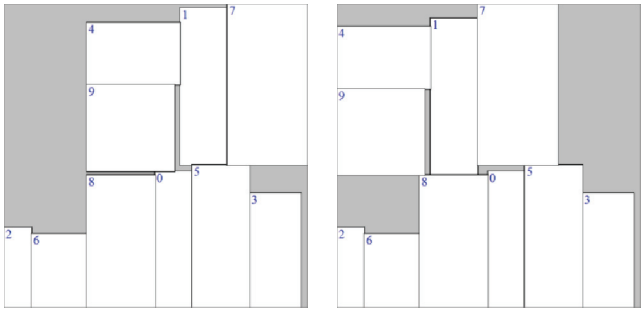
Step 3. If CS is better than BS (the dead space (DS) of CS is smaller than in BS), save CS as BS (BS = CS).

Step 4. If rotation is 270° (R = 270°), go to Step 5. If rotation is different than 270°, add 90° to current rotation value (R = R + 90°), save IS as CS (CS = IS) and rotate CS clockwise by R value, then go to Step 2.

Step 5. End of the algorithm, BS is the final solution given by constructional algorithm.

5.1 Blocks Sliding

To simplify the description, only the first quarter of the Cartesian coordinate system is used, where both coordinates (x, y) are positive, so the lower left corner of the floorplan is in the point (0, 0). Blocks sliding is the operation of moving blocks as close to the origin of the coordinate system as possible – down and left (two separate operations), only if there is enough space to move them – blocks cannot overlap after being moved. Sliding takes place recursively, beginning with the blocks, which are closest to the point of origin (from the left or from the bottom) – Figure 1.



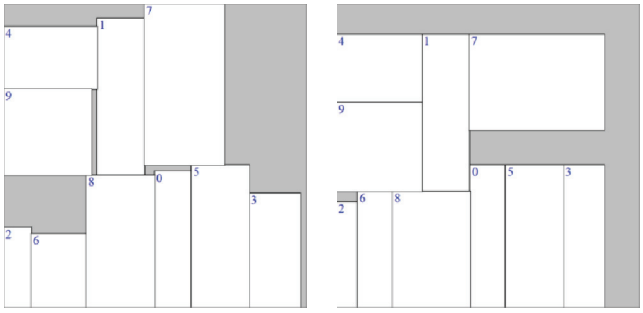
a) initial packing b) after sliding

Fig. 1. Example of blocks sliding to the left

5.2 Blocks Scaling

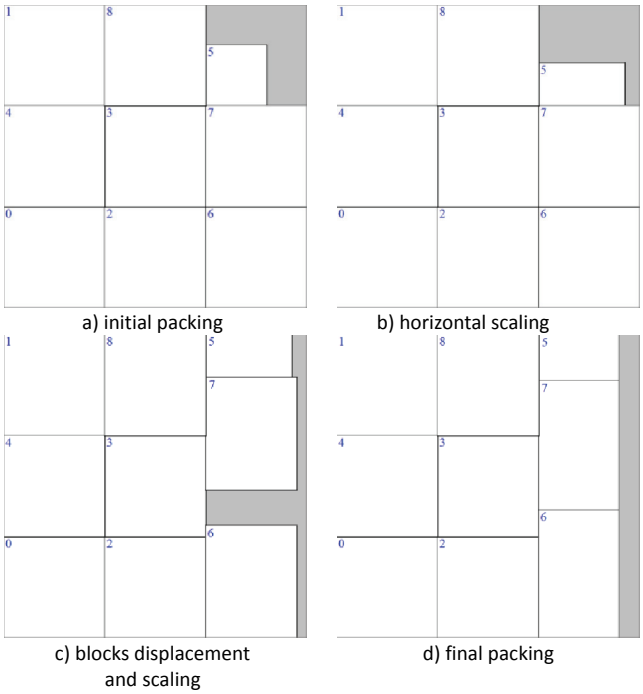
Blocks sliding operation eliminates empty spaces, which are on the left or below each of the blocks. *Soft-blocks* allow constrained modification of height and width. Using this property for each block with an empty space on the right, the algorithm changes the width or height (within the preset constraints – see (2)) to fill in the empty space if it is possible. This operation is called blocks scaling.

Figure 2 shows an example of blocks scaling. The height of the first two blocks in the lower left corner (blocks 2 and 6) was increased and the width was decreased (the overall area did not change). As a result, the empty space over blocks was filled by reducing the width of 2 and 6 and this enabled shifting other blocks from right side to the left.



a) initial packing b) packing after scaling

Fig. 2. Example of blocks scaling



a) initial packing b) horizontal scaling

c) blocks displacement and scaling d) final packing

Fig. 3. Example of scaling with displacement

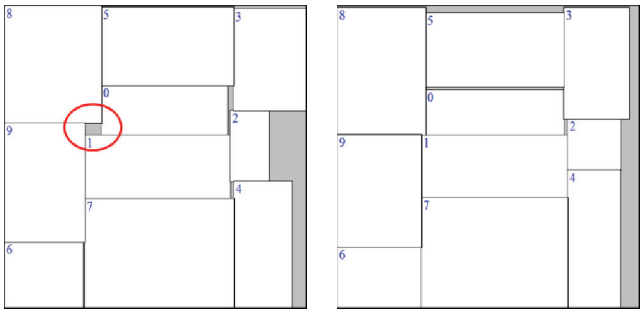
5.2 Scaling with Displacement

The two operations described above (sliding and scaling) can be used independently, but in some cases, using sliding and scaling independently does not give the desired results, e.g. the packing in Figure 3a is specific – the empty space is only around block 5. Therefore, sliding causes no changes and scaling would only turn the square block 5 into a rectangle – Figure 3b. In this case the desired result was not achieved – the area of the floorplan was not reduced.

Scaling with displacement is an 'intelligent' scaling performed as follows: the block with an empty space above is moved in up and scaled in order to increase height at the expense of width. The same action is performed also for each block which is below the moved one (Figure 3c). Then the algorithm starts to slide blocks – all of them are moved down as far as possible. The whole cycle is repeated until there are no empty spaces above the blocks (Figure 3d) or the height or width constraint is reached.

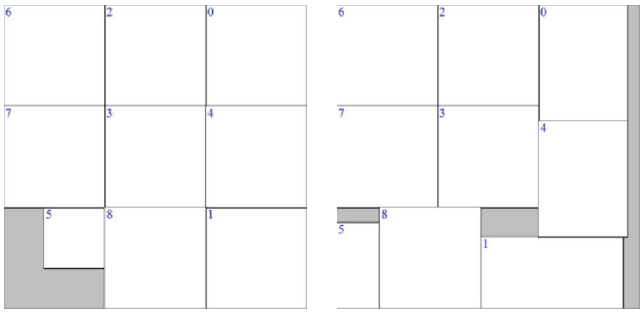
The above description refers to the operation of scaling with displacement for vertical orientation.

The algorithm also works the same for horizontal orientation.



a) initial packing b) packing after the elimination of the deadlocked empty spaces

Fig. 4. Example of eliminating deadlocked empty spaces between blocks 9, 8, 0, 1 and 7, 1, 2, 4



a) initial packing b) packing without rotation

Fig. 5. Example of optimization without rotation

5.4 Elimination of Deadlocked Empty Spaces

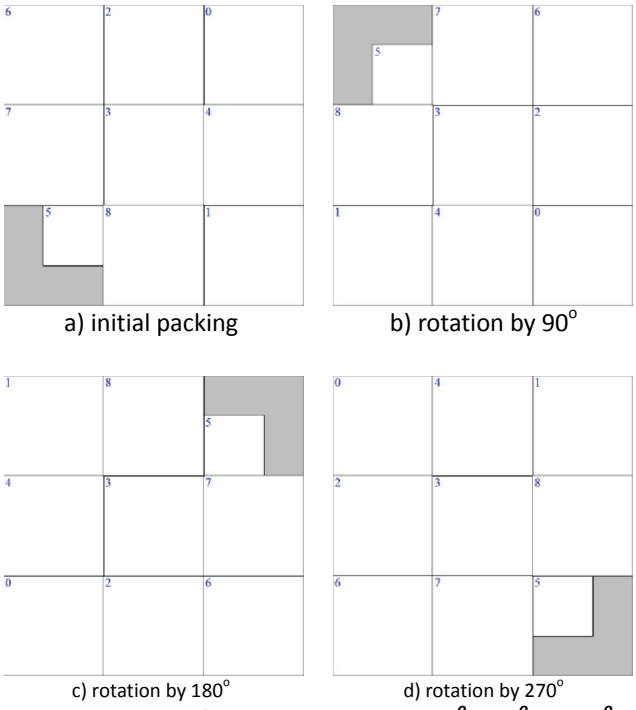
In some cases there are deadlocked empty spaces between four adjacent blocks, e.g. empty space between blocks 9, 8, 1 and 0 or 7, 1, 2 and 4 in Figure 4. Such a situation cannot be solved using the previous operations, because none of the blocks can be displaced or scaled. Any action would cause an overlap, which is not acceptable.

Optimization starts from spreading the blocks apart. For the example in Figure 4, the following operations will be performed: block 9 is expanded to the width of block 8 to fill in the empty space. At the same time, all blocks on the right of block 9 are shifted to the right by the resulting difference, so that they do not overlap (block 9 is a little different in its width from block 8 which is a consequence of other operations performed later). Analogically, in the second case, block 1 is expanded to the width of block 7 and block 2 is moved to the right so that it does not overlap with the expanded block 1.

5.5 Layout Rotation

The four previous operations do not solve the problem, when the empty space is located in the middle of the layout or close to the origin of coordinate system (see an example of the described packing – Figure 5). In this case, the empty space is moved as close to the upper right corner of floorplan as possible by rotating the whole layout. Rotation is made in clockwise direction, around the origin of coordinate system by 90° or a multiple of it.

Figure 5 shows that the process of packing optimization failed (four previous operations without rotation were used). The empty space decreased, but has not disappeared. For the same initial packing an optimal solution was obtained after rotation by 180° and using scaling and displacement operations (Figure 3d).



a) initial packing b) rotation by 90° c) rotation by 180° d) rotation by 270°

Fig. 6. Example of packing rotation by 90°, 180° or 270°

In the case, when the empty space is on the right or at the top of the floorplan, the results can be worse after rotating the layout. Therefore, the constructional algorithm is executed starting from initial and three rotated packings and chooses solution with the smallest value of DS as the final solution.

6. Experimental Analysis

Experiments were based on MCNC benchmarks available in the literature [22]. Four benchmarks were chosen (the value in the brackets is a number of blocks): *apte* (9), *xerox* (10), *hp* (11), *ami33* (33). For all of these, constraints were determined $k_{min} = 0.5$ and $k_{max} = 2.0$. For the experiments a computer was used with the following parameters: Windows 7, Intel(R) Core(TM) i7 CPU Q 840 @ 1.87GHz and 8 GB RAM.

Both simulated annealing algorithm (SA) (see section 4) and constructional algorithm (CA) (see section 5) require initial *sequence-pair*. This pair was generated randomly and was the same for both algorithms. The experiments were performed also for the algorithm, which combines the SA and CA (marked as SA+CA). In this algorithm, solutions obtained from the SA algorithm became the initial solutions for the CA algorithm.

10 different initial packings were randomly generated. For every one of these, we executed each algorithm separately: SA, CA and SA+CA. A constructional algorithm was run once for a single initial packing, because of its determinist operating. The SA algorithm was executed 50 times for every initial packing. The best from 50 results was chosen, which was considered as a final solution. For each trial the following parameters of simulated annealing were used

(determined during experiments): Tp = 400, ϵ = 0.001, α = 0.999.

Experiment 1

The simulated annealing algorithm requires *hard-blocks* to operate correctly. *Hard-blocks were obtained* from *soft-blocks* by setting three constant values of *k* = {*0.5; 1.0; 2.0*}, so the first stage of the algorithm is to create *hard-blocks* according to these three values. During the first experiment, results were obtained and compared for three algorithms (SA, CA, SA+CA) applied to 10 initial packings generated randomly and for three values of *k* ratio.

Bolded values in Table 1 are the best dead-space (DS) values for a particular benchmark, the same algorithm and three chosen values of *k*. The simulated annealing algorithm (SA), constructional algorithm (CA) and a combination of the two (SA+CA) are the most effective for blocks created for *k = 1.0*, that is for square blocks (4 out of 36 values turned out to be better for *k = 0.5*).

Table 1. Comparison of results obtained at different values of k for the simulated annealing algorithm (SA), constructional algorithm (CA) and a combination of the two (SA+CA). The table contains minimum, maximum and average percentage value of the DS criterion for 10 different initial solutions. BIS (Best Initial Solution) lines include values of the DS obtained for the initial solution which gave the best result

Benchmark		*k = 0.5*			*k = 1.0*			*k = 2.0*		
		SA [%]	SA +CA [%]	CA [%]	SA [%]	SA +CA [%]	CA [%]	SA [%]	SA +CA [%]	CA [%]
apte	*BIS*	49.83	6.93	7.80	12.22	11.93	1.32	16.15	15.72	5.29
	min	24.96	6.93	7.80	**4.93**	**1.94**	**1.32**	16.15	11.34	5.29
	avg.	44.02	19.97	20.90	**13.07**	**9.82**	**5.46**	39.95	21.67	30.56
	max	55.45	26.98	44.38	**20.07**	**15.18**	**8.18**	59.88	34.38	49.83
xerox	*BIS*	44.73	27.87	11.40	37.75	5.21	1.45	41.37	28.29	7.29
	min	25.63	16.67	11.40	**25.36**	**5.21**	**1.45**	25.65	11.36	7.29
	avg.	**35.75**	22.82	24.25	43.73	**16.63**	**15.45**	46.10	16.95	28.06
	max	**45.84**	**31.33**	49.99	58.65	32.55	**28.12**	66.77	32.56	53.35
hp	*BIS*	47.42	13.82	5.39	27.68	7.41	6.49	41.64	8.78	7.77
	min	28.55	13.53	**5.39**	**27.68**	**7.41**	6.49	20.97	8.78	7.77
	avg.	43.58	23.45	16.35	**43.23**	**18.90**	**12.16**	45.04	30.41	25.75
	max	59.66	45.50	34.14	69.87	**30.71**	**17.02**	**56.87**	56.87	46.34
ami33	*BIS*	131.71	30.54	11.55	92.23	40.10	10.33	136.89	38.31	19.88
	min	111.93	23.09	11.55	80.26	16.28	**10.33**	101.13	26.20	19.88
	avg.	145.69	31.25	23.42	94.33	23.62	**19.18**	136.68	32.75	33.45
	max	188.34	45.68	41.54	116.99	40.10	**32.77**	189.47	38.31	55.48

The problem is to find such a *soft-blocks* packing that minimizes the value of the DS criterion (3). The perfect result would be the value DS = 0%. A similar result was not obtained for any benchmark. The reason for this could be that the results are highly dependent on the results of the initial packing generating algorithm (in this case, packing is generated randomly). If the method of initial packing generation is changed, one would supposedly obtain a better result. Moreover, it is not determined that the best result produced by the SA algorithm is the most effectively minimized by the CA algorithm, e.g. for *k = 0.5* of the benchmark *xerox* the algorithm SA+CA improved the result from 44.73% to 27.87% (BIS line – see description of Table 1) and for the sample *apte* from 49.83% to 6.93%. To conclude, a worse initial solution can give a better final result (as in case of simulated annealing).

Experiment 2

In this experiment, is compared the average percentage value of DS and average time for the considered algorithms and the best algorithms known from the literature ([23], [1]). According to the results obtained from the previous experiment, *k = 1.0* was set to produce *hard-blocks* from *soft-blocks*.

Table 2. Comparison of the following algorithms: simulated annealing (SA), constructional algorithm to minimize block packing (CA) and combined algorithm (SA+CA) to the algorithm suggested in [23] (Lagrangian) and the branch and bound algorithm [1] (CompaSS). The column Initial Solution (IS) shows an average value of the DS for 10 initial packings

Benchmark		SA		SA + CA		CA		Lagrangian		CompaSS	
	IS [%]	DS [%]	t [s]	DS [%]	t* [ms]	DS [%]	t [ms]	DS [%]	t [s]	DS [%]	t [ms]
apte	99.19	13.07	19.22	9.82	19	5.46	25	0.54	53.0	0.75	50
xerox	49.42	43.73	23.68	16.63	146	15.45	20	0.40	71.6	0.00	35
hp	119.20	43.23	32.13	18.90	20	12.16	32	1.40	107.3	0.00	42
ami33	141.53	94.33	232.93	23.62	261	19.18	118	4.30	774.6	0.00	883

* Time of operation refers only to the constructional algorithm

According to data included in Table 2, it is seen that the constructional algorithm (CA) significantly improves (minimizes) both initial packing (DS decreased 18 times compared to the initial packing – sample file *apte*) and packing provided by the simulated annealing algorithm (DS decreased by 2.5 times – sample file *xerox*). The constructional algorithm is faster than the CompaSS algorithm on the same computer system, and much faster than simulated annealing based on Lagrange relaxation (results calculated on computer with processor Pentium III 600-MHz). It was observed that the execution time of the CA algorithm is up to 7.5 times less in comparison to, for example, the CompaSS algorithm. Unfortunately, effective values of the DS for solutions given by the CA algorithm are worse than those produced by algorithms from the literature. In conclusion, the considered constructional algorithm,

despite the fact that it gives solutions with worse DS, is faster than the other algorithms. This is the reason why it may be applicable to chips, which consist of a large number of elements where time is a considerable factor. It may also be used as an algorithm to improve packing results produced by an algorithm, which minimizes the length of interconnections [2].

7. Summary

The constructional algorithm CA described in this paper, significantly decreases the value of the DS criterion, and consequently, reduces the overall area of the floorplan for all tested benchmarks. It should be emphasized that the best results are obtained for those layouts that consist of square blocks in their initial packing ($k=1.0$). A great advantage of the algorithm is a short execution time and ability to start from any initial packing represented in Cartesian coordinate system. The proposed algorithm may be used to improve results obtained from other algorithms.

AUTHORS

Marcin Iwaniec*, Adam Janiak – Institute of IT, Automated Control Engineering and Robotics, Wrocław University of Technology, Z. Janiszewskiego 11/17, 50-372 Wrocław, Poland, marcin.iwaniec@pwr.wroc.pl, adam.janiak@pwr.wroc.pl

*Corresponding author

REFERENCES

[1] Chan H., Markov I., "Practical slicing and non-slicing block-packing without simulated Annealing", In: *Proc. Great Lakes Symposium on VLSI (GLSVLSI)*, 2004, pp. 282–287.

[2] Chen T.C., Chang Y.W., Lin S.C., "IMF: Interconnect-driven multilevel floorplanning for large-scale building-module designs. In: *Proc. ICCAD'05, ACM/IEEE*, 2005, pp. 159–164.

[3] Chen T.C., Chang Y.W. Modern floorplanning based on B*-tree and fast simulated Annealing", *IEEE Trans. on CAD*, vol. 25, no. 4, 2006, pp. 637–650.

[4] Chen S., Yoshimura T., "Fixed-outline floorplanning: block-position enumeration and a ew method for calculating area costs", *IEEE Trans. Computer – Aided Des. Integrated Circuits Syst.*, vol. 27, no. 5, 2008, pp. 858–871.

[5] Chen J., Zhu W., Ali M.M., "A hybrid simulated annealing algorithm for nonslicing VLSI floorplanning", *IEEETrans. Syst., Man, and Cybernetics, Part C: Appl. Rev*., vol. 41, no. 4, 2011, pp. 544–553.

[6] Chiang C.W., "Ant colony optimization for VLSI floorplanning with clustering constraint", *J. Chin. Inst. Ind. Eng.*, vol. 26, no. 6, 2009, pp. 440–448.

[7] Hayes J.P., Murray B.T., *Testing ICs: getting to the core of the problem*, IEEE Computer Magazine, no. 11, 1996, pp. 32–38.

[8] Itoga H., Kodama C., Fujiyoshi K., "A graph based soft module handling in floorplan", *IEICE Trans. Fundam. Electron. Commun. Comput. Sci*., E88-A(12), 200, 5, pp. 3390–3397.

[9] Janiak A., Kozik A., Lichtenstein M., "New perspectives in VLSI design automation: deterministic packing by Sequence Pair", *Annals of Operations Research*, Springer Netherlands, 179, 2010, pp. 35–56

[10] Kang M., Dai W.-M., "General floorplanning with l-shaped, t-shaped and soft blocks based on bounded slicing grid structure". In: *Design Automation Conference 1997. Proceedings of the ASP-DAC '97. Asia and South Pacific*, 1997, pp. 265–270.

[11] Kirkpatrick S., Gelatt C., Vecchi M., "Optimization by Simulated Annealing", *Science*, no. 220(4598), 1983.

[12] Liu J., Zhong W.C., Jiao L.C., Li X., "Moving block sequence and organizational evolutionary algorithm for general floorplanning with arbitrarily shaped rectilinear blocks", *IEEE Trans. Evol. Comput.*, vol. 12, no. 5, 2008, pp. 630–646.

[13] Ma Y., Hong X., Dong S., Cai Y., Cheng C.-K., Gu J., *Floorplanning with abutment constraints based on corner block list*, Integration, the VLSI Journal, 31(1), 2001, pp. 65–77.

[14] Murata H., Fujiyoshi K., Nakatake S., Kajitani Y., "VLSI module placement based on rectangle-packing by the sequence pair", *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, vol. 15, no. 12, 1996.

[15] Otten R.H.J.M., van Ginneken L.P.P.P., "Floorplan design using simulated annealing". In: *Proc. Intl. Conf. on CAD*, 1984, pp. 96–98.

[16] Rebaudengo M., Reorda M.S., "GALLO: genetic algorithm for floorplan area optimization", *IEEE Trans. Comput. – Aided Des. Integrated Circuits Syst.*, vol. 15, no. 8, 1996, pp. 943–951.

[17] Valenzuela C.L., Wang P.Y., "VLSI Placement and Area Optimization Using a Genetic Algorithm to Breed Normalized Postfix Expressions", *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, 2002, pp. 390–401.

[18] Wang H.Y., Hu K., Liu J., Jiao L.C., „Multiagent evolutionary algorithm for floorplanning using moving block sequence", *IEEE Congr. Evol. Comput.,* 2007, pp. 4372–4377.

[19] Wong D.F., Liu C.L., "A new algorithm for floorplan design". In: *Proc. Design Autom. Conf.*, 1986, pp. 101–107.

[20] Xiaoping T., Ruiqi T., Wong D.F., "Fast evaluation of sequence pair in block placement by longest common subsequence computation", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 20, no. 12, 2001, pp. 1406–1413.

[21] Xiong E.S.J., Wong Y.-C., He L., "Constraint driven i/o planning and placement for chip-package co-design". In: *Proc. Asia and South Pacific Design Autom. Conf.,* 2006, pp. 207–212.

[22] Yang S., *Logic synthesis and optimization benchmarks*, Microelectronics Center of North Carolina, Research Triangle Park, N.C., Tech., 1991.

[23] Young F., Chu C., Luk W., Wong Y., "Handling soft modules in general nonslicingfloorplan using lagrangian relaxation", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 5, 2001, pp. 687–692.