Improving 3D-Floorplanning Using Smart Selection Operations in Meta-Heuristic Optimization

Artur Quiring, Markus Olbrich and Erich Barke
Institute of Microelectronic Systems
Leibniz Universität Hannover
Appelstraße 4, 30167 Hannover, Germany
Email: {artur.quiring, markus.olbrich, erich.barke}@ims.uni-hannover.de

***Abstract*—In 3D-Floorplanning even more than in 2D-Floorplanning new objectives, e.g. temperature, TSV-Planning or IR-Drop are considered. This increases the complexity of the problem formulation and, therefore, of the optimization algorithm, dramatically. Apart from some analytical approaches, simulated annealing based algorithms (SA) are widely used for 3D-Floorplanning. To increase the solution quality of classical SA, a common approach is to adapt the selection operations, improving local search. While previous work proposes selection operations which consider mostly one single design issue (e.g. temperature or fixed-outline), we propose a comprehensive multiobjective floorplan optimization methodology (smart SA) which is capable of efficiently considering several objectives and constraints (area, wirelength, fixed-outline, maximum number of TSVs and maximum temperature) at the same time. For the objectives and constraints we present simplified analysis models. Experimental results show that our extended SA algorithm outperforms the classical one and finds valid solutions where classical SA fails.**

I. Related Work

2D- and 3D-Floorplanning have been a well studied field in physical design of integrated circuits. Apart from the datastructures, modelling the relations between blocks, the optimization algorithms, optimizing the block arrangement regarding a cost function, are of major interest. Here due to the small number of blocks, their different aspect ratios and the considered constraints, simulated annealing based algorithms (SA) are still the most promising heuristics. There have been as well some analytical approaches [1], [2], [3], [4] to the floorplanning problem. In [1] for example a forced directed algorithm could be successfully demonstrated and in [2] the floorplanning problem was formulated and solved as a linear integer program. Nevertheless, SA is the preferred optimization algorithm due to its simplicity of integrating new objectives to the floorplanning problem and its high solution quality.

Ever since SA is used for floorplanning, there has been always adaptions to the classical SA algorithm to improve its runtime and solution quality. For example one can use different temperature schemes. Recently there have been reported successful temperature schemes in [5], [6] and [7]. In [7] the authors propose a three-phase SA algorithm, which in the first phase does normal random search to explore the solution space and to not get trapped into a local minimum in the very beginning. In the second phase it uses a pseudo-greedy approach to converge towards a local minimum and in the third phase it raises the temperature of the SA to facilitate the hill-climbing stage at the end of the SA run. All the mentioned contributions have in common that the SA temperature curve is split into phases, where every phase has a different start temperature, cooling schedule and optionally a different parameter set for the cost function.

Due to the increase of newly considered objectives in 3D-Floorplanning, e.g. thermal-aware planning [5], power ground planning [8], TSV planning [9], IR-Drop computation [10], changing the alteration operations to better guide the SA, appears to be a good approach. In [5] the SA alteration operations (move, swap, rotate) are extended with adapted local operations, which only change the floorplan little, leading to better performance of the SA in the latter phase, when acceptance rate of uphill moves is low. To create valid floorplans, complying with the fixed-outline constraint, the following approach is proposed in [11]. Firstly, a new objective is added to the cost function, representing the fixed-outline overflow. Additionally a slack-based method is proposed. It identifies the blocks, violating the fixed-outline constraint. When there is a violation the SA algorithm moves the blocks, responsible for the overflow, to a more promising position. This approach produces in most cases overflow-free results, while classical SA fails.

II. Motivation

A good temperature scheme can speed up the SA for classical 3D-Floorplanning, however, it is not clear how this approach can improve 3D-Floorplanning, considering many objectives and constraints. Therefore, especially when considering constraints, [11] motivates the adaption of the SA alteration operations, which appears to be a better approach. Unfortunately, there are no SA alteration operations, handling multiple constraints simultaneously. But, since the number of constraints in 3D-Floorplanning is likely to grow, we need an optimization algorithm, which is able to handle multiple constraints and to optimize multiple optimization objectives simultaneously.

III. Our Contribution

We propose an approach, which we refer to as smart SA. It considers all design issues at a time, guiding the SA algorithm to find a valid solution, complying with all constraints and optimizing the objectives simultaneously. For this purpose it uses adapted SA selection operations, depending on additional analysis values for the design issues. The computation of this values should be fast and sufficient to noticeably improve the local search of the SA. We present simplified analysis methods for all considered design issues.

The paper is organized as follows. Section IV defines the problem, we are solving. Section V discusses in detail the proposed analysis methods for fixed-outline, wirelength, number of TSVs and temperature. The smart SA is shown in section VI. Experimental results are given in section VII and the paper ends with a conclusion in section VIII.

IV. Problem Formulation

In most other publications the number of TSVs and the maximum temperature are treated as optimization objectives. Since they conflict with the optimization of wirelength and area, the designer usually has to find a trade-off between all optimization objectives, extensively tuning the weights in the cost function. In contrast, our optimizer considers the number of TSVs and the maximum temperature as hard constraints. The designer can set them to fixed values, depending on the additional area and yield due to the TSVs and the available thermal budget.

The optimization problem we are solving in this paper can be stated the following. Given a fixed-outline, a maximum temperature and a maximum number of TSVs, find a 3D-Floorplan which complies with this constraints, while reducing the wirelength and the area.

V. Analysis of design issues

To better guide the SA, we compute for every design issue specific analysis values, we refer to as SA-Information. The used 3D-Floorplan datastructure impacts the corresponding methods and is shortly described in the following.

We use the B^*-tree [12] datastructure, which has to be proven a good candidate for 2D-Floorplanning. The B^*-tree is an ordered binary tree. Every node i in the tree represents a block b_i in the packing. The computation of the packing from a B^*-tree is explained in the example, depicted in Fig. 1 and Fig. 2. Firstly, the block b_B of root B is placed at the origin. Then the B^*-tree is traversed in depth-first order, visiting the left child first. For a left child i of a parent j, b_i is placed to the right of b_j, with $x_{b_i} = x_{b_j} + w_{b_j}$, where x_{b_j} is the lower left x coordinate and w_{b_j} the width of b_j. In Fig. 1 b_C is placed to the right of b_B. For a right child i of a parent j, b_i is placed above b_j with $x_{b_i} = x_{b_j}$. In Fig. 1 b_D is placed above b_A. In both cases the lower left y coordinate depends on the already placed blocks and can be efficiently determined, using a contour data structure.

To represent a 3D-Floorplan we use the multi-level B^*-tree, a vector of B^*-tree, where every B^*-tree corresponds to a different layer. Based on the 3D-Floorplan datastructure we describe the analysis methods in the following subsections.

A. Fixed-outline

To estimate for every block the contribution to fixed-outline overflow, we compute longest path values for them, similar to [13]. The aforementioned properties of the B^*-tree demands two different algorithms for the longest path values; one for the x direction and one for the y direction. The algorithms have been taken from [13] and adapted to B^*-tree. Since the examples from [13] also apply to the B^*-tree they are reused here for clarification.

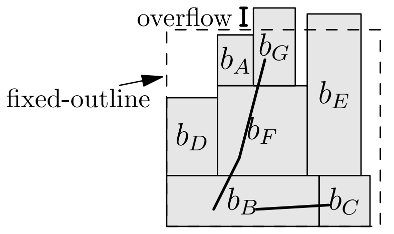


Fig. 1: Longest path values of block B in x and y direction.

1) Longest Path Values in x Direction: The longest path values in x direction can be easily determined by post order traversal of the B^*-tree. For the traversal, at first, the left edge and its subtree and then the right edge and its subtree are visited. A demonstration of the method is shown in Fig. 2, where $l_{x,i}$ denotes the longest path value for a node i in x direction, x_{h,b_i} denotes the upper x coordinate (right side of rectangle) of block b_i and the numbering of the edges represents the processing order of the edges. To determine $l_{x,i}$ for a node i the maximum of the longest path values of its children nodes $l_{x,j} \mid j \in i_{Children}$ and the upper x coordinate of its corresponding block b_i are compared. The calculation rule $l_{x,i} = max\{x_{h,b_i}, \{l_{x,j} \mid j \in i_{Children}\}\}$ is used. Considering for example the B^*-tree and its packing shown in Fig. 2 and Fig. 1, it can be seen that $x_{h,b_E} > x_{h,b_G}$ and $x_{h,b_E} > x_{h,b_F}$, resulting in $l_{x,F} = x_{h,b_E}$ for node F. To reduce a possible fixed-outline overflow in x direction, a node i with a high $l_{x,i}$ should be moved.

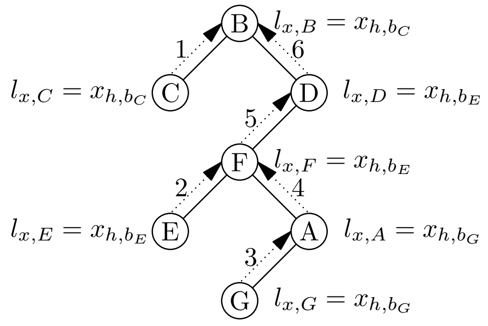


Fig. 2: B^*-tree containing longest path information for x direction.

2) Longest Path Values in y Direction: Different from the x direction, the computation of the longest path values in y direction can not be directly determined from the B^*-tree. Instead, a graph G_{dep} shown in Fig. 3 describing the dependencies between blocks in y direction is needed. G_{dep} is computed during the packing method in the following way. At first, G_{dep} consists of a virtual node S which represents the x axis. Subsequently, every time a block b_i is placed, the contour data structure is used to determine its y coordinate, thereby getting a set of blocks M_{above} which are totally or partially covered by b_i in the y direction. Node i is then added as a child of j with $max\{y_{h,b_j}, b_j \in M_{above}\}$ in G_{dep}. Regarding the example in Fig. 1 and in Fig. 3 when b_F is placed, M_{above} contains only b_B, therefore, an edge between F and B is added in G_{dep}. When b_E is placed, M_{above} contains b_C and b_B. Since $y_{h,b_C} = y_{h,b_B}$ both edges (C, E) and (B, E) are added. Once G_{dep} is computed it can be traversed in post-order-like manner, indicated by the edge numbering in Fig. 3. Similar to the x direction, the calculation rule $l_{y,i} = max\{y_{h,b_i}, \{l_{y,j}, j \in i_{Children}\}\}$ is used, where y_{h,b_i} denotes the higher y coordinate (top side of rectangle) and $l_{y,i}$ denotes the longest path value in y direction of node i.

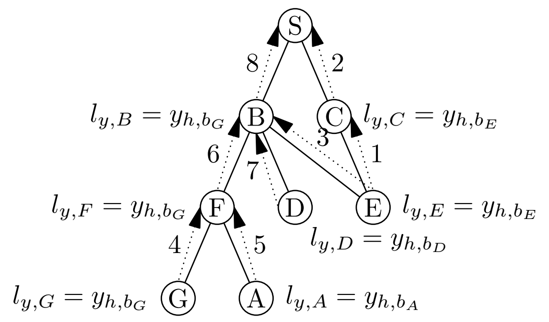


Fig. 3: Dependency graph G_{dep} containing longest path information for y direction.

After computing $l_{x,i}$ and $l_{y,i}$ for all blocks b_i, the longest path values $l_i = l_{x,i} + l_{y,i}$ are calculated and used as SA-Information for the fixed-outline constraint. The greater l_i the greater the probability to reduce the fixed-outline overflow.

B. Wirelength

To estimate the wirelength improvement I_{b_i} of moving a block b_i, we propose the following method. In Fig. 5a the position of block b_i and all nets $n_i \in N_{b_i}$ connected to b_i are shown. Summing up the *HPWLs* of all $n_i \in N_{b_i}$, represented by dashed rectangles, gives us H_{b_i}. Let $H_{b_{i,+}}$ be the sum of all *HPWLs* of $n_i \in N_{b_i}$ after moving b_i to another position. Estimating the maximum wirelength improvement, when moving b_i, can be done, computing an optimal position (x_{opt}, y_{opt}), so that $H_{b_{i,+}}$ is minimal:

$$H_{b_{i,min}} = \min H_{b_{i,+}} \tag{1}$$

As a first estimation for (x_{opt}, y_{opt}) the mass center point of all blocks $b_j \in n_j \mid n_j \in N_{b_i}$ can be used. Unfortunately, with the mass center point the quadratic wirelength is minimized and not the HPWL. To get an idea for the difference, Fig. 4 depicts the optimal placement position P_1, in terms of HPWL minimization, for a block, belonging to the nets n_1, n_2 and n_3. P_1 is always found by our method. The misplaced position determined when computing the mass center point is P_2. The impact of this misplacement on the overall wirelength optimization is further investigated in section VII. To compute

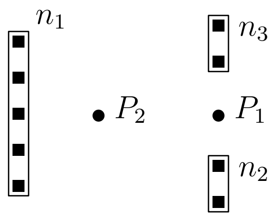
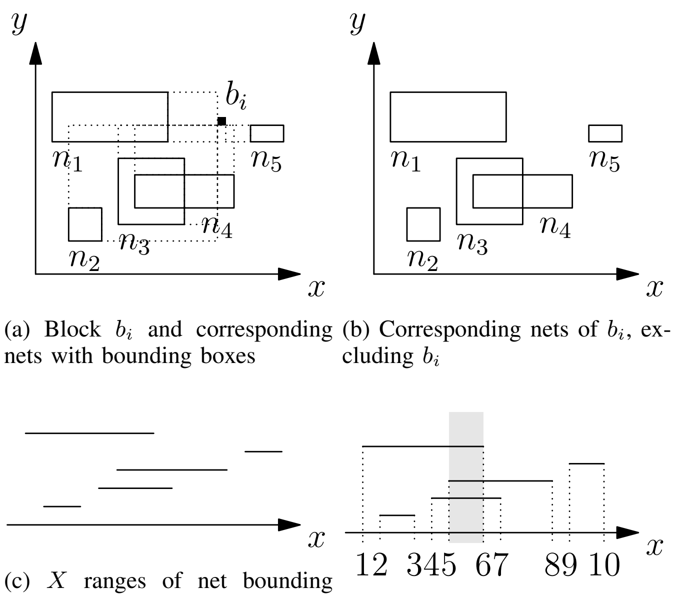


Fig. 4: Optimal Position P_1 and mass center point P_2



(a) Block b_i and corresponding nets with bounding boxes
(b) Corresponding nets of b_i, excluding b_i
(c) X ranges of net bounding boxes
(d) Range for optimal position

Fig. 5: Computing position for minimal HPWL

the optimal position, the set of nets $N_{\bar{b_i}}$ of b_i excluding b_i itself is needed, shown in Fig. 5b. Now the problem can be split into two independent problems, considering the x ranges and the y ranges separately. Fig. 5c shows the x ranges of the bounding boxes of $N_{\bar{b_i}}$. Placing b_i at some x coordinate and adding the x coordinate to the x ranges, leads to the following increase:

$$x_{inc} = H_{x,b_{i,+}} - H_{x,\bar{b_i}} \tag{2}$$

For the optimal x coordinate, x_{inc} has to be minimal. To be minimal, b_i has to be placed in the range $\hat{r}_x$, shown as shaded rectangle in Fig. 5d. To determine $\hat{r}_x$, all corner points of the x ranges have to be sorted in ascending order, represented by the numbering in Fig. 5d. $\hat{r}_x$ now consists of the two center points of the sorted corner points, 5 and 6 in the example. The same computation is done for the y ranges, resulting in $\hat{r}_y$. Now the maximum improvement is:

$$I_b = x_{inc} + y_{inc} \tag{3}$$

and the optimal position is:

$$(x_{opt}, y_{opt}) = (x, y) \mid x \in \hat{r}_x, y \in \hat{r}_y \tag{4}$$

I_{b_i} is used as the SA-Information for the wirelength. The greater I_{b_i} the greater the probability to reduce the wirelength, when moving b_i.

C. Number of TSVs

In order to guide the SA, reducing the number of TSVs, we propose the following heuristic. Compute for every block the total reduction of TSVs when moving that block to another layer. Given an exemplary floorplan with three layers. A block b is placed on layer one and connected to the nets n_1 and n_2, depicted in Fig. 6. The other blocks in n_1 are all placed on layer two and the blocks in n_2 are placed on layer two and three. This floorplan has in total three TSVs, one for n_1 and two for n_2. Moving b to layer two would decrease the number of TSVs for n_1 by one and for n_2 also by one, resulting in one TSV for the modified floorplan. If instead b is moved to layer three, the number of TSVs is one for n_1 and one for n_2 which in total is two. So we get for b the TSV reduction values $(0, 2, 1)$ for the three layers. Moving b to layer two gives us the highest reduction in the number of TSVs, hence, we use this value as the SA-Information for TSVs. The greater this value the greater the probability to reduce the number of TSVs. The drawback of this approach is its tendency to place

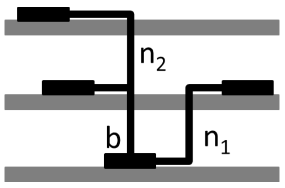


Fig. 6: Block b with corresponding nets n_1 and n_2

all modules only on one layer, maximally reducing the number of TSVs. Therefore, we slightly adapt the SA-Information for TSVs. Since we know the position of the blocks, we compute for every layer the total area of blocks located on it. Dividing this area by the total area of all blocks, we get the area fraction $area_{layer}$ for a particular layer. To equally distribute the blocks to the layer, $area_{layer}$ should be approximately the total area divided by the number of layers. Having a smaller $area_{layer}$ means the layer is sparse and there should be more blocks placed on it. With this information we weight the SA-Information for TSVs, whereas for blocks which are located on a sparse layer, the SA-Information value is reduced and for blocks located on a crowded layer it is increased. That way, we favour the movement of blocks from crowded layers to sparse layers in the optimization later on.

D. Temperature

For thermal analysis the model described in [5] is used. It enables fast computation of temperatures for discrete areas (tiles), sufficient to identify hot spots in the floorplan. Furthermore, it gives a good approximation of temperature relations between different areas of the floorplan. To bias the simulated annealing towards reduction of maximum temperature, for every block b_i a temperature information t_{b_i} is computed in the following way. b_i might intersect with several tiles and for every tile a temperature value is computed in the thermal analysis model, depicted in Fig. 7. t_{b_i} is the sum of the temperatures of all intersecting tiles of b_i. A b_i with a high t_{b_i} was placed most likely next to a hot spot. Therefore, moving b_i near a block b_j with low t_{b_j} can reduce the maximum temperature significantly. Analog to the TSV information, this approach also tends to place all modules on one layer, most reducing the temperature. Therefore, the temperature information, equally to the TSV information, is weighted with the $area_{layer}$ value, presented in section V-C.

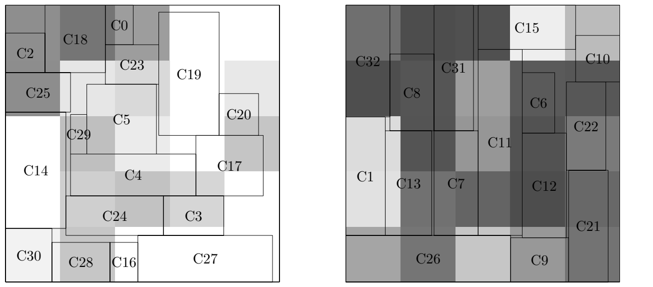


Fig. 7: Floorplan with temperature map (left = layer 1; right = layer 2; dark tiles = high temperature)

VI. Smart SA method

The smart SA algorithm is based on classical SA. Though, it has a different selection procedure for determining the next block to be altered. It uses the analysis values from the methods, presented in section V and combines them to a discrete distribution, depicted in Fig. 8. In the discrete distribution every block has a probability assigned to it. That way, blocks 2, 9 and 12 for example have a high probability to be altered, due to its higher chance of improving the design issues (maximum temperature, maximum number of TSVs, wirelength and fixed-outline overflow).

The analysis values have all different dimensions (e.g. temperature and wirelength), making them impossible to be combined directly in the discrete distribution. Therefore, in a first step the values of every analysis method are normalized to a range $[0, 1]$, preserving the relations. In a second step a simple heuristic is used, summing up for every block its normalized values.

Using the discrete distribution for the selection, instead of a uniform distribution (used in classical SA), should improve the local search characteristics and guide the SA more efficiently through the solution space. But, when adapting the SA alteration operations, the random character of SA should be preserved, to allow escaping from local minimum. In [11] this is done, using a random selection (with a uniform distribution) in every nth SA step. In contrast smart SA assigns every block a positive probability to be altered, still allowing to escape

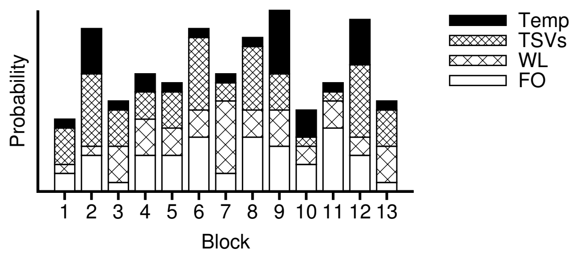


Fig. 8: Example for a discrete distribution containing analysis values

TABLE I: Comparison of mass center point computation and our method.

	Mass center point		Our method	
	Area (mm^2)	Wire (mm)	Area (mm^2)	Wire (mm)
ami33	1.34	78.85	1.33	69.89
ami49	41.48	1079.65	41.30	1014.18
n100	0.221	227.46	0.220	221.69
n200	0.218	432.27	0.217	416.15
n300	0.335	586.87	0.334	559.00
Avg. $(\Delta_{\%})$	0	0	-0.42	-5.64

from local minimum and making the aforementioned random selection steps superfluous.

VII. Experimental Results

Our multi-constraint-driven 3D-Floorplanner is implemented in C++ on Linux. All results were computed, using the MCNC and GSRC benchmarks.

At first, to show the quality of our analysis model for wirelength, we compared it with the approach where the mass center point is used.

To the best of our knowledge there are no publications, considering maximum temperature, maximum number of TSVs and fixed-outline as constraints. Hence, in a second test, we compared our smart SA with classical SA. And, finally, we modified the selection operation in smart SA, to compare different strategies with our proposed smart SA.

A. Comparison Of Different Wirelength Analysis Methods

The presented method for wirelength in section V-B should be more suitable to the HPWL metric and, therefore, should produce better results compared to the frequently-used computation of the mass center point [11], which tries to minimize the quadratic wirelength.

Firstly, we run our optimization algorithm, using our wirelength analysis method to compute the optimal position (x_{opt}, y_{opt}). To focus on wirelength optimization, the results were determined for a 2D-Floorplan, neglecting number of TSVs and temperature. Afterwards we used the mass center point as the optimal position, keeping all other parameters the same. The results in Tab. I show that our wirelength analysis method produces on average 5 % less wirelength with about equal area for the benchmarks, due to the more accurate analysis model for HPWL.

B. Smart SA Compared To Classical SA

To show the efficiency of our approach we compared it to the classical simulated annealing. For both methods we used the same initial temperature, temperature scheme and number of SA steps. The initial probability of accepting an uphill move was set to 0.8. The weights of the cost function were also set to the same values.

In Fig. 9 and Fig. 10 experimental results for the benchmark $n100$ show the success rate of finding a valid solution (100 runs for every configuration), when maximum TSV number (plotted on the x-axis) and maximum die temperature (plotted on the y-axis) are set to fixed values. The fixed-outline is determined, using the same width and height and a maximum of 20 % whitespace for $n100$. For a maximum die temperature of 240 and for a maximum number of TSVs of 550, smart SA for example has a success rate of 49 %, where classical SA does not find any valid solution. The results for the other benchmarks confirm with the results of $n100$, hence, we omitted them for the sake of clarity.

Depending on the benchmark, the overall runtime of our optimization tool was in between a few seconds and 15 minutes. The average increase in runtime for smart SA, compared to classical SA, was about 30%, due to the additional computation of all analysis values.

Overall, it can be seen that classical SA mostly fails to comply with the constraints. Even when using twice as much SA steps in classical SA the results did not improve significantly. Smart SA performs much better in meeting the constraints, making it a good candidate for 3D-Floorplanning with multiple objectives and especially with multiple constraints.

Because our approach searches the solution space more efficiently it produces overall better results. We observed that in the beginning of SA our method reduces costs very fast, since it prefers move operations, reducing all objectives at a time and that way following a higher gradient in the solution space than classical SA. Furthermore, we observed, that in the end of SA, when temperature is low and uphill moves are very unlikely to be accepted, our method has a better chance of improving the result than classical SA.

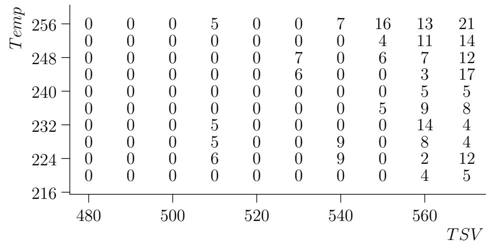


Fig. 9: Success rate for $n100$ using classical SA

C. Comparison Of Smart SA and Greedy Random Approach

Instead of using the discrete distribution for the selection in smart SA, a greedy approach similar to the one reported in [11] can be used.

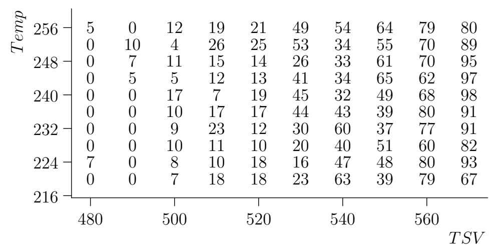


Fig. 10: Success rate for $n100$ using smart SA

We tested it, modifying the selection operation, so that the block with the highest probability (most promising block) is always chosen. To escape from local minimum, the method selects in every 4th step a random block. The results for $n100$ are shown in Fig. 11. It can be seen that this methodology performs very badly in our case, even worse than classical SA. In a second test, we used our approach with a random selection in every 4th step without the greedy part from the first test. The random step is again to not get trapped into a local minimum. But, because in smart SA every block has already a probability higher zero to be selected, the additional random selection steps are counterproductive, leading to worse results, shown in Fig. 12.

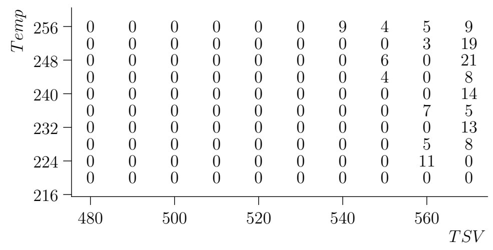


Fig. 11: Success rate for $n100$ using smart SA with greedy selection and intermediate random selection

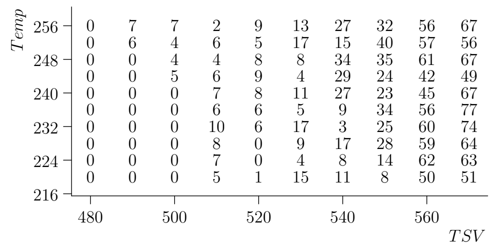


Fig. 12: Success rate for $n100$ using smart SA with intermediate random selection

VIII. Conclusion

The success of our approach depends on the analysis functions. The results for the wirelength show, that carefully selecting an accurate analysis function to guide the SA towards a certain design issue, is very important. Usually the more accurate the models the better the results but also the higher the overall runtime of the algorithm. In this work we presented simplified analysis functions for all considered design issues. Furthermore our proposed optimization algorithm dramatically improved the success rate of finding a valid floorplan with little runtime overhead, compared to classical SA.

In our ongoing research we are trying to further improve the analysis functions. Furthermore, we are investigating different schemes to compute the discrete distribution function. Some design issues might be more important than others. Weighting them higher, not only in the cost function, but also in the discrete distribution function might improve the results.

References

[1] P. Zhou, Y. Ma, Z. Li, R. Dick, L. Shang, H. Zhou, X. Hong, and Q. Zhou, "3d-staf: scalable temperature and leakage aware floorplanning for three-dimensional integrated circuits," in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, 2007, pp. 590 –597.

[2] J.-G. Kim and Y.-D. Kim, "A linear programming-based algorithm for floorplanning in vlsi design," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 22, no. 5, pp. 584 – 592, May 2003.

[3] Y. Zhan, Y. Feng, and S. S. Sapatnekar, "A fixed-die floorplanning algorithm using an analytical approach," in *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '06. Piscataway, NJ, USA: IEEE Press, 2006, pp. 771–776. [Online]. Available: http://dx.doi.org/10.1145/1118299.1118477

[4] J. Cong, M. Romesis, and J. Shinnerl, "Fast floorplanning by look-ahead enabled recursive bipartitioning," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 9, pp. 1719 –1732, 2006.

[5] J. Cong, J. Wei, and Y. Zhang, "A thermal-driven floorplanning algorithm for 3d ICs," in *ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 306–313.

[6] L. Xiao, S. Sinha, J. Xu, and E. Young, "Fixed-outline thermal-aware 3d floorplanning," in *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, 2010, pp. 561 –567.

[7] T.-C. Chen and Y.-W. Chang, "Modern floorplanning based on b*-tree and fast simulated annealing," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 4, pp. 637 – 650, april 2006.

[8] C.-W. Liu and Y.-W. Chang, "Power/ground network and floorplan cosynthesis for fast design convergence," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 4, pp. 693 –704, april 2007.

[9] M.-C. Tsai, T.-C. Wang, and T. Hwang, "Through-silicon via planning in 3-d floorplanning," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 8, pp. 1448 –1457, aug. 2011.

[10] Q. Zhou, J. Shi, B. Liu, and Y. Cai, "Floorplanning considering ir drop in multiple supply voltages island designs," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 4, pp. 638 –646, april 2011.

[11] S. Adya and I. Markov, "Fixed-outline floorplanning: enabling hierarchical design," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 11, no. 6, pp. 1120 – 1135, 2003.

[12] Y.-C. C. Yao-Wen, Y. chih Chang, Y. wen Chang, G. ming Wu, and S. wei Wu, "B*-trees: A new representation for non-slicing floorplans," in *Proc. DAC*, 2000, pp. 458–463.

[13] A. Quiring, M. Lindenberg, M. Olbrich, and E. Barke, "3d floorplanning considering vertically aligned rectilinear modules using T*-tree," in *3D Systems Integration Conference (3DIC), 2011 IEEE International*, 31 2012-feb. 2 2012, pp. 1 –5.