Self-Aligned Double-Patterning Aware Legalization

Hua Xiang Gi-Joon Nam Gustavo Tellez Shyam Ramji
IBM T.J. Watson Research Center
Yorktown Heights, NY USA
{huaxiang, gnam, tellez, ramji }@us.ibm.com

Xiaoqing Xu
Univ of Texas at Austin
Austin, TX USA
XiaoqingXu.Austin@utexas.edu

***Abstract*—Double patterning is a widely used technique for sub-22nm. Among various double patterning techniques, Self-Aligned Double Patterning (SADP) is a promising technique for good mask overlay control. Based on SADP, a new set of standard cells (T-cells) are developed using thicker metal wires for stronger drive strength. By applying this kind of gates on critical paths, it helps to improve the design performance. However, a mixed design with T-cells and normal cells (N-cells) requires that T-cells are placed on circuit rows with thicker metal, and the normal cells are on the normal circuit rows. Therefore, a placer is needed to adjust the cells to the matched circuit rows. In this paper, a two-stage min-cost max-flow based legalization flow is presented to adjust N/T gate locations for a legal placement. The experimental results demonstrate the effectiveness and efficiency of our approach.**

***Index Terms*—SADP, Network Flow, Legalization**

I. Introduction

Although the technology has scaled into sub-22nm and below, the manufacturing still relies on a 193nm (ArF) wavelength light source. The difficulties in forming patterns on a wafer are becoming more and more challenging. Various techniques have been developed. One promising approach is Self-Aligned Double Patterning (SADP). SADP forms a wafer image by single exposure followed by sidewall spacer processes [12] for better overlay controllability.

Deploying the SADP technique, a set of standard cells is designed. The new set cells have the same height as the normal cells, but they use thicker metal tracks for stronger drive strength such that they could provide better performance. This kind of gates is called T-gates. (For convenience, the normal gates are referred to as N-gates.) T-gates are inserted along critical paths for delay reduction. Although the cell height of T-gates is unchanged, the circuit rows hosting these new cells must use a thicker metal. This leads to a mixed-cell design.

Given a design, each circuit row is marked as a "normal" row or a "thick" row. For example, there are three circuit rows in Fig. 1. $Row1$ and $Row3$ are the normal rows with three tracks, while $Row2$ is the row with thicker tracks. Accordingly, $g1 \sim g6$ are the normal gates, and $g7 \sim g10$ on $Row2$ are the T-gates. To simplify the presentation, N-row refers to the normal circuit rows and T-row to the rows with thick tracks.

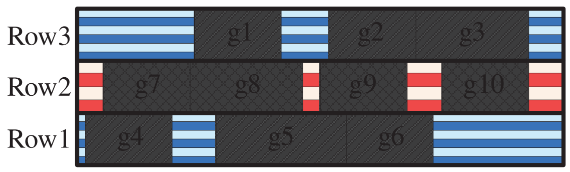

Fig. 1. A mixed design with N-gates and T-gates. $g1 \sim g6$ are N-gates, and $g7 \sim g10$ are T-gates. T-gates must be placed on circuit rows with thicker tracks, such as $Row2$.

Given a design with mixed N-gates and T-gates, the placement must satisfy the gate location constraints, i.e., N-gates are on N-rows and T-gates are on T-rows. For convenience, this constraint is called as N/T matching constraint.

If there are no N/T matching constraints, the general placer can be applied for a legal placement since all cells and rows have the same height. However, it is very likely that N-gates are on T-rows, and T-gates are on N-rows. And the mismatch must be corrected for a legal SADP-aware placement.

SADP-aware Legalization Given a mixed N/T-cell design, some gates are fixed. A legal placement without N/T matching constraints is provided. The objective is to adjust the gate locations, such that the T-gates are placed on the T-rows, the N-gates are placed on the N-rows, and the fixed gates are kept untouched. At the same time, minimizing the changes of the gate location is also critical.

[9] presented a floorplan implementation with mixed 8T and 12T cells. But 8T and 12T cells have different cell heights, and the implementation is to group cells into blocks so that each block deploys one technology node gates. In this work, the N/T-cells use the same technology and N/T matching constraints are row-based.

The general placement legalization is to resolve overlaps among gates, and its quality directly affects the design performance. Many works [3]–[7], [10], [11], [13], [14] were published to address this issue. However, the N/T matching constraints are hard constraints, and cannot be easily integrated into these approaches. Among these works, the network flow is also widely used [3], [7], [10]. [7] formulated each gate and each region to a graph node. For large designs, the corresponding graph is huge. Furthermore, the flow network cannot model the discrete gate sizes, and the mapping between flow solution and gate movement is hard to interpret.

In this paper, a two-stage min-cost max-flow based legalization flow is presented to adjust N/T gate locations for a legal SADP-aware placement. The major contributions of this paper include the following:

- A novel min-cost max-flow formulation is proposed which nicely captures the N/T matching constraints. The graph construction also supports gate movement minimization.
- An efficient flow realization method is developed to simplify the mapping from network flows to gate movement.
- A new gate/group swapping technique is introduced to address the gate assignment in congested regions. Meanwhile, the swapping groups can be smoothly fitted in the proposed flow network.
- An efficient two-stage N/T gate assignment flow is presented. The two stages use different methods to estimate region capacity (i.e., the number of gates that can be inserted into the region). It guarantees the convergence of the flow.

The rest of the paper is organized as follows: Section 2 presents the two-stage gate assignment flow. Section 3 presents the algorithm details. Experimental results are presented in Section 4, and Section 5 concludes the paper.

II. Two Stage Legalization Flow

In this section, a two-stage iterative legalization algorithm is presented to relocate "mismatched" gates to the matched circuit rows.

In a design, there might be fixed objects such as pre-placed gates and IPs. Based on these fixed objects, each circuit row is partitioned into block-free regions. As shown in Fig 2, the solid regions are occupied by fixed objects, and they partition the circuit rows into discontinuous regions.

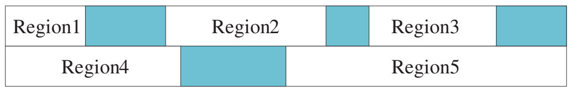

Fig. 2. The solid blue blocks are fixed objects. They partition the two circuit rows into 5 regions.

The input is a legal placement without N/T matching constraints. Although there is no overlap among gates, some gates might be placed on the incompatible circuit rows. For each region, gather the mismatched gates. The target is to relocate "mismatched" gates to their compatible circuit rows. Meanwhile, guarantee that each region has no overflows, i.e., the total gate width is less than the region width. Since the N/T gates/rows have no overlap, the N-gate and T-gate assignment can be handled separately.

After identifying the mismatched gates, a two-stage min-cost max-flow based algorithm is applied to assign them to the matched circuit rows.

At each stage, a flow network is built, and the min-cost max flow algorithm is applied to assign gates to matched regions. However, different gates might have different widths, and the flow network cannot group flows. How to map the flow to gate assignment becomes a challenging problem. In [3], [7], a gate is sliced into a unit, and the flow may only represent a partial gate, and a special step of flow realization is required.

Contrary to the previous work, each gate is used as one flow unit. At the first stage, the average gate width is used to estimate region capacity, i.e., how many gates can be inserted in the region. The average gate width gives a good estimation on how many gates can be assigned. But still, during the flow mapping, it's possible that some gates cannot fit in the assigned region. These unassigned gates will get allocated in the second stage.

In the second stage, the gates are bucketed based on gate widths. The gates with the same/close width are grouped in the same bucket, and the buckets are processed one by one. For each bucket, pick the gate with the maximum gate width and use this value to estimate region capacity. After applying the min-cost max-flow algorithm, the mapping from the flow to the region is guaranteed.

Finally, after assigning gates to regions, there might be overlaps among gates. Region placement can be applied to resolve the gate overlaps. Region placement is a well-studied problem [4], [5], and many approaches such as single-row or 1D linear placement can efficiently legalize the gates in regions. The details of region placement are omitted in this paper. Fig 3 summarizes the whole flow.

III. Min-Cost Max-Flow Based Region Assignment

In this section, we present the network flow based region assignment algorithms for the two stages separately.

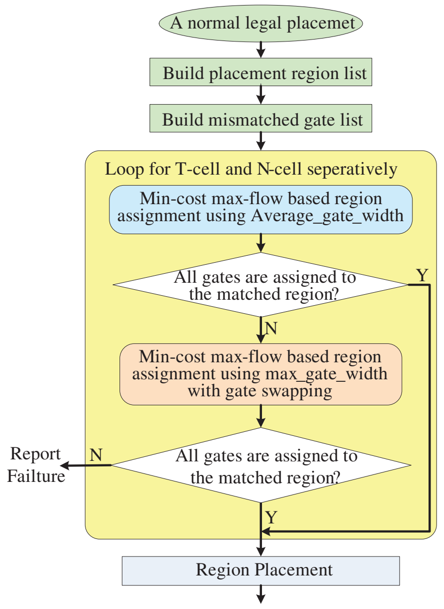
A normal legal placemet
Build placement region list
Build mismatched gate list
Loop for T-cell and N-cell seperatively
Min-cost max-flow based region assignment using Average_gate_width
All gates are assigned to the matched region?
Y
N
Min-cost max-flow based region assignment using max_gate_width with gate swapping
All gates are assigned to the matched region?
N
Report Failture
Y
Region Placement

Fig. 3. Mismatched gate assignment flow

A. First Stage: Flow Network with Average Gate Width

In this section, we propose an iterative network flow based approach to assign mismatched gates to their compatible regions. The algorithm is summarized as follows.

```
Algorithm Flow_based_region_assignment_ave_gatewidth()
For (iter=1; iter ¡ iter_threshold; iter ++)
1.   Calculate ave_mismatch_w of mismatched gates;
2.   Build network graph with (row_range, x_range);
3.   Apply min-cost max-flow algorithm;
4.   Derive region assignment with exact gate width;
5.   If the corresponding region has no overflow
5.1    Then assign gates directly to the corresponding region
5.2    Else assign largest width gates first if they can be fitted
5.   Update mismatched gate list;
6.   If the mismatched gate list is empty, then break;
7.   Increase row_range or x_range;
8.   If row_range and x_range have reached the threshold,
9.   Then break;
```

In this formulation, the average gate width of mismatched gates is used to estimate region capacity. The flow network $G = (V_f, E_f)$ is constructed as follows.

(1) $V_f = \{s,t\} \cup V_g \cup V_{inr} \cup V_{outr}$, where s is the source node, and t is the sink node. $V_g = \{g_i | i = 1..n\}$ is the mismatched gate node set. $V_{inr} = \{r_i | i = 1..m\}$ and $V_{outr} = \{p_i | i = 1..m\}$ are the in-node and out-node sets for row regions, respectively.

(2) $E_f = E_s \cup E_t \cup E_r \cup E_g$, where
$E_s = \{(s, g_i) | g_i \in V_g, i = 1..n\}$;
$E_t = \{(p_i, t) | p_i \in V_{outr}, i = 1..m\}$;
$E_r = \{(r_i, p_i) | r_i \in V_{inr}, p_i \in V_{outr}, i = 1..m\}$;
$E_g = \{(g_i, r_j) | g_i \in V_g, r_j \in V_{inr}$, if g_i can assign to region r_j.$\}$

(3) Edge Capacity:
$e \in E_s, U_s(e) = 1$;
$e \in E_t, U_t(e) = \infty$;

$e \in E_r, U_r(e) = (R_{width} - R_{used})/ave\_mismatch\_w;$
$e \in E_g, U_g(e) = 1;$

(4) Edge Cost:
$e \in E_s, C_s(e) = 0;$
$e \in E_t, C_t(e) = 0;$
$e \in E_r, C_r(e) = 0;$
$e \in E_g, C_g(e) = wirelength\_cost;$

Each mismatched gate is represented by a node g_i. Also each region is represented by a capacitized node. The capacity is the number of gates that can be inserted into the region. It is estimated as $(R_{width} - R_{used})/ave\_mismatch\_w$, where R_{width} is the total region width, R_{used} is the total width of gates already in the region.

Since the flow network doesn't support capacitized nodes, a pair of nodes (r_i, p_i) is used to represent a region. As shown in Fig 4, Q_i is a region node with a capacity of Cap and a cost of $Cost$. Q_i is represented by two nodes r_i and p_i. An edge (r_i, p_i) connects r_i and p_i. Cap and $Cost$ are the capacity and cost of (r_i, p_i), respectively.

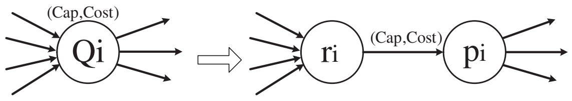

Fig. 4. Node splitting. A capacitized node Q_i is converted to two nodes (r_i, p_i).

All edge cost is zero except the edges from gates to region in-nodes. The edge cost of (g_j, r_i) is the wirelength change when the gate g_j is moved to region r_i.

At this step, the gate overlap constraint is not considered. The wirelength change is based on the minimum gate location changes.

Without loss of generality, assume that the circuit rows are horizontal rows, and the gate center (g_x, g_{row}) is used to represent the location of the gate g. For regions, $(r_{xlow}, r_{xhigh}, r_{row})$ denotes one region. If g_x is within $[r_{xlow}, r_{xhigh}]$, then g_x is unchanged. If $g_x < r_{xlow}$, $g_x = r_{xlow}$; otherwise, $g_x = r_{xhigh} - g_{width}$, where g_{width} is the width of the gate. Fig 5 shows the x dimension calculation when a gate is moved to a region. The total wirelength change is $\Delta g_x + row_{height} \cdot \Delta rows$.

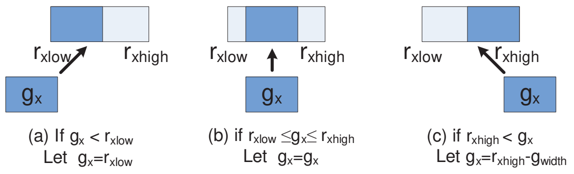
r_{xlow}
r_{xhigh}
g_x
(a) If $g_x < r_{xlow}$ Let $g_x=r_{xlow}$
r_{xlow}
r_{xhigh}
g_x
(b) if $r_{xlow} \leq g_x \leq r_{xhigh}$ Let $g_x=g_x$
r_{xlow}
r_{xhigh}
g_x
(c) if $r_{xhigh} < g_x$ Let $g_x=r_{xhigh}-g_{width}$

Fig. 5. x-location calculation when a gate is moved to a region.

Fig 6 gives one example of the flow network graph. In this example, there are three mismatched gates g_1, g_2 and g_3, and four regions. Each region is represented by a pair of nodes $(r_i, p_i) i = 1..4$. g_1 can be assigned to $region1$ and $region2$, g_2 can move to $region1$, $region2$ or $region3$, and g_3 can be inserted to $region3$ and $region4$. The edge cost from g_i to r_j is the wirelength change if g_i is moved to r_j.

After constructing the flow graph, the min-cost max-flow algorithm is applied. If there is a flow on edge (g_i, r_j), then assign g_i to region r_j. However, the region capacity is estimated based on the average width of mismatched gates. It's very likely that some regions cannot fit all assigned gates. The strategy is to sort assigned gates and assign the gates with largest width first. In general, a gate with a small width is relatively easy for relocation. The algorithm is summarized as follows.

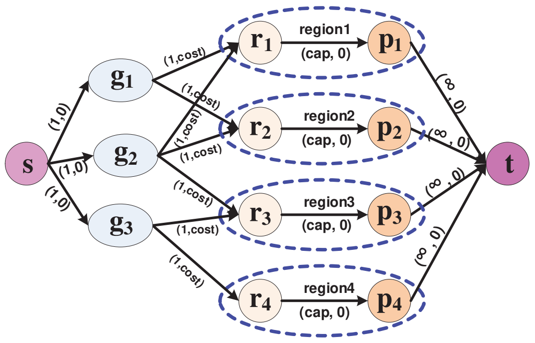

Fig. 6. A flow network graph example

```
Algorithm Region_Assignment_from_Ave_Gatewidth_Flow()
For each edge $e = (g, r) \in E_g$
1.   If there is no flow on $e$, continue;
2.   Add $g$ into the candidate list of $r$;

For each region $r$
1.   If the width of all candidates $\leq (R_{width} - R_{used})$
2.   Then assign all candidates to $r$;
3.   Else sort candidates with decreasing gate width;
4.            loop candidates from the largest width
5.                  If a candidate can fit in $r$, then assign it to $r$
```

After assigning some gates to regions, the mismatched gate list should be updated. And the next iteration will work on the updated mismatched gate list. Furthermore, each iteration will increase the search space, i.e., increasing row_range or x_range or both. The loop stops if the mismatched gate list is empty, which means all gates get legal region assignment. Or the number of iterations has reached the given threshold. Also if the row_range and x_range are larger than the specified threshold, the iterations stop.

The core part of the region assignment algorithm is the min-cost max-flow algorithm, which is a classical problem, and several polynomial algorithms are available [1], [2], [8]. If the double scaling algorithm in [2] is used, the time complexity can be bounded by $O(|V_f| \cdot |E_f| \cdot \log |V_f|)$ where $|V_f|$ is the number of nodes in the flow network, and $|E_f|$ is the number of edges. It is easy to see that $|V_f|$ is linearly bounded by the mismatched gates and the number of regions. And $|E_f|$ is bounded by $O(|V_f|^2)$.

B. Second Stage: Flow Network with Max Gate Width

When the average mismatched gate width is used to estimate region capacity, it's very likely that the flow network finds all the flows, but the gates cannot be assigned. This affects the convergence of the algorithm. To avoid this issue, the max gate width is used in the second stage.

The mismatched gates have different widths. First, the mismatched gates are partitioned into buckets. The gates in the same bucket have the same or very close width. Let max_gate_width be the maximum gate width of the bucket. For each bucket, apply the min-cost max-flow based region assignment algorithm. But this time, max_gate_width is used to estimate region capacity. Since gates in one bucket have the same or very close sizes, max_gate_width can pretty accurately reflect the region capacity. Meanwhile, the maximum gate width guarantees the one-to-one mapping from the flow to gate assignment.

In some congested areas, a gate might not be able to find a feasible location or have to be assigned to a far away legal location. Fig 7 (a) gives an example. In Fig 7 (a), the gate A is a N-gate, but it is assigned to a T-row. However, the empty space in all regions is not big enough to host A, i.e., A cannot find a legal location. To address this issue, swapping groups are introduced into the network graph so that the gates have better chances to be placed close to their original locations.

1) Swapping Groups: For a mismatched gate, search its neighborhood regions. If two gates are exchangeable, they should have the same gate width. It's very likely that few gates in the neighborhood have the exact same gate width as the given gate. So, consider gate groups, i.e., a gate group whose total width is the same as the given gate. Then the gate is exchangeable with the group. For convenience, this kind of gate groups are called as swapping groups.

Suppose there are K gates in one region r. The total number of non-empty subgroups is $2^K - 1$. However, it is not necessary to explore all combinations. The groups are to swap with a gate, and it is not good to exchange one gate with a large amount of gates. Therefore, we limit the number of gates inside each group. Usually, 2 or 3 gates per group are good enough. Then the total subgroup is bounded by $O(K^2)$ or $O(K^3)$. The algorithm is summarized in Select_Swap_Group. The number of gates in one group is controlled by the parameter $gnum$. gwt is the gate width threshold. g_{width} refers to the gate width of a group. r_{empty} is the empty space of region r.

Algorithm Select_Swap_Group(int $gnum$, int gwt, region r)
```
1.  Enumerate groups whose gate number < $gnum$;
2.  For each group $g$,
3.      If ($g_{width} > gwt$), remove $g$ from the list;
4.      If (($g_{width} < gwt$)&&($r_{empty} \leq gwt - g_{width}$)),
5.          remove $g$;
6.      $g_{empty} = gwt - g_{width}$;
7.  Sort group list based on ave_group_movement_cost;
8.
9.  SelectedGroups = { }
10. SelectedGates = { }
11. UsedEmptySpace = 0
12. For $g$ in the group list
13.     If (UsedEmptySpace +$g_{empty} < r_{empty}$)
14.         If (all gates in $g$ are not in SelectedGates)
15.             Add $g$ into SelectedGroups;
16.             Add all gates in $g$ into SelectedGates;
17.             UsedEmptySpace $+=$ $g_{empty}$;
18.
19. Return SelectedGroup
```

Since the groups are used to exchange with another gate, the total gate width of a group has to be the same as the given width. If the group gate width (g_{width}) is larger than the given width (gwt), the group should be dropped. On the other hand, if the group gate width is smaller than the given width, the empty space is added to meet the width requirement.

After creating all potential groups, they are sorted based on their movement_cost. Given a region, the group_movement_cost is the total wirelength change when all gates in a group are moved to the region.

For a mismatched gate, its searching space is determined by row_range and x_range. Once the searching space is determined, we use ave_group_movement_cost to estimate the group movement cost. ave_group_movement_cost is the average of group_movement _cost when trying all regions in the searching space.

When selecting swapping groups, except the group gate width requirement, it also requires that no gates are shared by any two selected groups. Note that if a gate belongs to multiple groups, it means that a gate is assigned to multiple locations, and it is not feasible for placement. Therefore, SelectedGates is used to record gates which are already picked. Only when none of the gates in the group appear in SelectedGates, the group can be added to SelectedGroups. Also some groups may include empty space. Hence the total empty space in SelectedGroup cannot exceed the empty space of the region. UsedEmptySpace is used to check the feasibility of the empty space in a group. Accordingly, after adding one group into SelectedGroups, SelectedGates and UsedEmptySpace should be updated as shown in Line 15-16.

The following is an example to illustrate the selection of swapping groups. Assume there are five gates $g_1^{10}, g_2^8, g_3^5, g_4^5, g_5^2$ in one region. (The lower number is the gate index. The upper number is the size of the gate.) Assume the empty space in the region is 5, and the limit of the gates in one group (g_{num}) is 2. The target gate width (gwt) is 8. If the gate width of a group is smaller than 8, some empty space has to be added to make 8. For convenience, we denote p^i as an empty space with size i. The total number of groups under the gate limit is $C_5^2 + C_5^1 = 10 + 5 = 15$ groups, i.e., $\{g_1^{10}\}$, $\{g_2^8\}$, $\{g_3^5, p^3\}$, $\{g_4^5, p^3\}$, $\{g_5^2, p^6\}$, $\{g_1^{10}, g_2^8\}$, $\{g_1^{10}, g_3^5\}$, $\{g_1^{10}, g_4^5\}$, $\{g_1^{10}, g_5^2\}$, $\{g_2^8, g_3^5\}$, $\{g_2^8, g_4^5\}$, $\{g_2^8, g_5^2\}$, $\{g_3^5, g_4^5\}$, $\{g_3^5, g_5^2, p^1\}$, and $\{g_4^5, g_5^2, p^1\}$.

But with the width constraint, only 6 groups are left. $\{g_2^8\}$, $\{g_3^5, p^3\}$, $\{g_4^5, p^3\}$, $\{g_5^2, p^6\}$, $\{g_3^5, g_5^2, p^1\}$, $\{g_4^5, g_5^2, p^1\}$.

Also the empty space of the region is 5. So $\{g_5^2, p^6\}$ is pruned. Sort the rest of the groups based on their ave_group_movement_cost. Suppose the sorted group list is $\{g_2^8\}$, $\{g_3^5, p^3\}$, $\{g_4^5, p^3\}$, $\{g_3^5, g_5^2, p^1\}$, and $\{g_4^5, g_5^2, p^1\}$.

$\{g_2^8\}$ and $\{g_3^5, p^3\}$ are first picked. The leftover empty space is only 2. $\{g_4^5, p^3\}$ cannot be selected since it consumes 3 empty space. $\{g_3^5, g_5^2, p^1\}$ cannot be picked either because $\{g_3^5\}$ is already in the selected groups. Finally, $\{g_4^5, g_5^2, p^1\}$ is selected. So the final swapping groups are $\{g_2^8\}$, $\{g_3^5, p^3\}$ and $\{g_4^5, g_5^2, p^1\}$.

2) Flow Network with Swapping Groups: After identifying swapping groups, the gates in the groups are removed from their original locations, and will get re-assigned together with the mismatched gates using the flow network. The algorithm is summarized as follows.

Algorithm Flow_based_region_assignment_with_swap_groups()
```
1.  Build gate buckets based on gate widths;
2.  Sort buckets with decrease gate widths;
3.  For each bucket
3.1.    Let the max gate width of the bucket be $G_w$
3.2.    For (iter=1; iter¡iter_threshold; iter ++)
3.2.1     For each gate in the bucket
3.2.1.1       Identify swapping groups
3.2.2     Build graph with (x_range, row_range, groups)
3.2.3     Apply min-cost max-flow algorithm;
3.2.4     Assign gates based on the flow;
3.2.5     Remove assigned gates from the bucket;
```

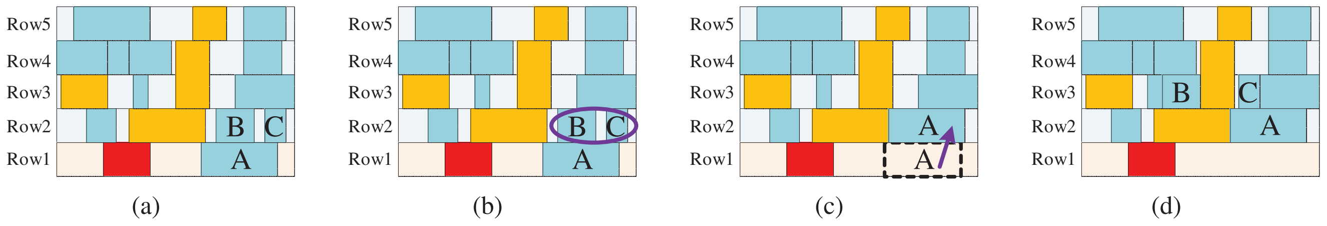

Fig. 7. (a) The N-gate A is placed on a T-row, but it cannot be moved to any other regions; (b) A swapping group $\{B, C\}$ is identified; (c) Gate A is moved to the original location of $\{B, C\}$; (d) Gate B and C are assigned in later iterations.

```
3.2.6   Insert unassigned group gates into the buckets;
3.2.7   If the bucket is empty, break;
3.2.8   Increase row_range or x_range;
3.2.9   If row_range and x_range reach the threshold,
3.2.9.1   break;
```

3) Second Stage Gate Assignment: Each group is represented by a node. If a region is within the allowable assignment area of the group, one edge is created to connect the group node and the region node. Fig 8 shows an example. In this example, there are two mismatched gates $g1$ and $g2$. Based on $g1$ and $g2$, two swapping groups $group1$ and $group2$ are identified in the search area of $g1$ and $g2$. $Group1$ can be assigned to $region2$, $region3$, and $region4$, while $group2$ can be assigned to $region3$ and $region4$. Edges are added to connect group nodes to the region nodes.

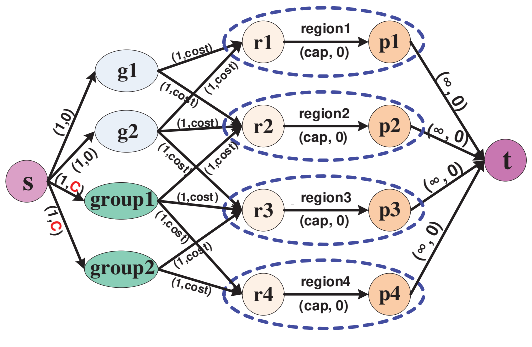

Fig. 8. A flow network with swap candidate groups

In order to give the mismatched gates a higher priority, a relatively large cost is assigned to the edges from the source node to the swapping group nodes. Since the edge cost from the source node to the mismatched gate nodes is zero, the mismatched gates have better chances to get assigned first. Meanwhile, the region capacity is calculated by the maximum gate width. It guarantees the mapping from the flow to the gate region assignment. In other words, if there is a flow from one gate to a region, the gate can be inserted into the region for sure. This greatly enhances the convergence of the whole flow.

All the mismatched gates are partitioned into buckets based on their gate widths. Gates in the same bucket has the same or similar widths. For each bucket, first apply Select_Swap_Group on each mismatched gate to identify swapping groups as Line 3.2.1 in the algorithm Flow_based_region_assignment_with_swap_groups. A network graph including the swapping groups is constructed. Based on the min-cost max-flow solution, the gates are assigned to regions. Since the maximum gate width is used, the gates guarantee to fit in the calculated regions.

After one iteration of gate assignment (Line 3.2 in the algorithm), it is possible that some mismatched gates or swapping groups are not assigned. For mismatched gates, they will be handled in the future iterations with a larger search window. But for swapping groups, it is not necessary to allocate all gates of one group to one region. Instead, each gate in one swapping group is inserted to a bucket according to the gate width. This gives more opportunities to have gates assigned. As the example in Fig 7 (b), a swapping group $\{B, C\}$ is identified. In the first iteration, gate A is assigned, while $\{B, C\}$ cannot find a legal location. At the end of the first iteration, gates B and C are inserted separately into different buckets since they have different gate widths. Later they will get assigned when the corresponding buckets are processed. In Fig 7 (d) shows that B and C are inserted in different regions.

IV. Experimental Results

The proposed algorithms are implemented in C++, and integrated them into a physical-synthesis system. All the test cases were derived from the industrial designs, and they were tested on a Linux workstation (2.66GHz). T-Gates are the gates on thicker metal tracks, and N-Gates are the normal gates. The T-rows are periodically assigned with a T/N ratio of 1:4.

TABLE I
Design Information

Test	ChipSize	Total	T-Gates		FixedGates	
	(um^2)	Gates	Num	Percent	Num	Percent
Design1	235.52x115.20	21122	3655	17.30%	7414	35.10%
Design2	163.84x115.20	17923	2680	14.95%	4842	27.02%
Design3	153.60x230.40	36934	6415	17.37%	4626	12.53%
Design4	209.92x220.16	53755	7796	14.50%	6073	11.30%
Design5	163.84x186.88	31459	3952	12.56%	4486	14.26%
Design6	199.68x135.68	12026	828	6.89%	3312	27.54%
Design7	235.52x261.12	58637	7411	12.64%	6772	11.55%
Design8	307.20x240.64	89345	3037	3.40%	10757	12.04%
Design9	222.72x281.60	62742	7253	11.56%	6471	10.31%
Design10	204.80x138.24	44059	2094	4.75%	2861	6.49%
Design11	163.84x320.00	55564	4504	8.11%	4694	8.45%
Design12	133.12x143.36	16173	1660	10.26%	3564	22.04%
Design13	97.28x435.20	41436	2937	7.09%	5843	14.10%

The algorithms are compared against a greedy approach. For the greedy method, all invalid gates are first sorted based on their gate widths. Then process one gate each time starting from the gate with the largest width. When one gate is processed, search in its neighborhood. The neighborhood is a rectangular region. The original gate location is the center and the rectangle size is bounded by the x/row threshold. For each feasible location in the neighborhood rectangle, calculate the gate movement cost using the same cost function as the one in SADP-placer. The gate will be placed in the min-cost location. Table I lists the design information.

Table II reports the TWL comparison results. Original TWL is the total wire length of the input placement. TWL is the wire length after region assignment and region placement. For all designs, the wire length degradation is very limited with the proposed flow. On average, the total wire length increase is only 2.8%, while the greedy approach increases the wire length by 29.4%. Table III and Table IV list the comparison results on location changes. $LocChange$ is the total distance change between the original location and the final location. For both $AverageLocChange$ and $MaxLocChange$, the proposed approach outbids the greedy method.

Fig 9 shows a piece of design3. Fig 9 (a) is the original design, and (b) is the SADP-aware placement. The green gates are fixed objects, the red ones are T-gates, and the rest are the N-gates.

TABLE II
TWL COMPARISON

Test	Original	Greedy		SADP	
		TWL	IncPer	TWL	IncPer
Design1	3820553	4580032	19.88%	3903583	2.17%
Design2	2316156	3064601	32.31%	2389950	3.19%
Design3	8291986	10733032	29.44%	8550358	3.12%
Design4	9804721	14063315	43.43%	10186344	3.89%
Design5	5108608	6963491	36.31%	5355238	4.83%
Design6	2432549	2796101	14.95%	2471108	1.59%
Design7	13046947	17229130	32.05%	13287153	1.84%
Design8	17270284	20841892	20.68%	17683274	2.39%
Design9	14919907	18690843	25.27%	15203571	1.90%
Design10	6511903	9037735	38.79%	6863351	5.40%
Design11	9814958	13584808	38.41%	10173947	3.66%
Design12	3356038	4126053	22.94%	3428483	2.16%
Design13	8180818	10020393	22.49%	8328011	1.80%
Ave			29.42%		2.81%

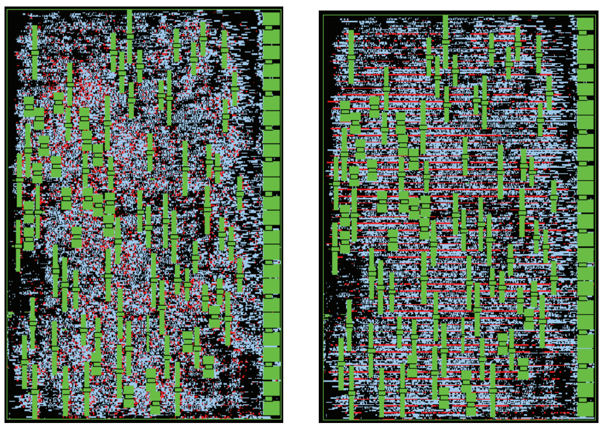
(a) $Design3$ input layout (b) $Design3$ SADP placement

Fig. 9. Placement before and after SADP placement

V. CONCLUSION

In this paper, we present a two-stage min-cost max-flow based SADP region assignment flow to reassign mismatched gates to the appropriate circuit rows. The experimental results demonstrate the effectiveness and efficiency of our approach.

TABLE III
COMPARISON RESULTS ON AVERAGE LOCATION CHANGE AND RUNTIME

Test	Average			Runtime (s)	
	Greedy	SADP	DiffPer	Greedy	SADP
Design1	1918.66	1789.41	-6.74%	0.82	1.04
Design2	2001.57	1608.70	-19.63%	1.81	0.9
Design3	3592.65	2073.43	-42.29%	5.79	3.36
Design4	4135.80	1945.19	-52.97%	6.85	3.57
Design5	3043.34	1778.76	-41.55%	4.96	3.15
Design6	1814.65	1394.88	-23.13%	0.59	0.96
Design7	4033.29	1545.62	-61.68%	7.1	4.72
Design8	2051.68	1623.52	-20.87%	17.63	6.85
Design9	3419.30	1488.15	-56.48%	8.06	5.14
Design10	2926.38	2052.84	-29.85%	4.66	3.32
Design11	3347.01	1554.32	-53.56%	5.23	4.76
Design12	2706.16	1716.09	-36.59%	1.46	1.08
Design13	2480.95	1368.35	-44.85%	3.41	2.68
Ave			-41.45%	5.41	3.19

TABLE IV
COMPARISON RESULTS ON GATE MAX LOCATION CHANGE

Test	T-Gates			N-Gates		
	Greedy	SADP	Diff	Greedy	SADP	Diff
Design1	39680	15680	-60.48%	20080	19680	-1.99%
Design2	42240	16080	-61.93%	20960	16880	-19.47%
Design3	60240	29200	-51.53%	24000	16960	-29.33%
Design4	104400	23840	-77.16%	46320	14960	-67.70%
Design5	49360	22480	-54.46%	36720	16800	-54.25%
Design6	30240	9680	-67.99%	19840	16720	-15.73%
Design7	81040	25280	-68.81%	61680	13760	-77.69%
Design8	38880	16640	-57.20%	45600	23360	-48.77%
Design9	86800	18240	-78.99%	47360	19440	-58.95%
Design10	71200	18000	-74.72%	64240	27200	-57.66%
Design11	61440	20960	-65.89%	75840	13360	-82.38%
Design12	60240	18480	-69.32%	32320	17840	-44.80%
Design13	54160	19200	-64.55%	36400	14400	-60.44%
Ave			-67.46%			-56.46%

REFERENCES

[1] R. K. Ahuja, A. V. Goldberg, J. B. Orlin, and R. E. Targan, Finding minimum-cost flows by double scaling, Mathematical programming 53, pp. 243-266, 1992.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Network Flows, Prentice Hall, 1993.

[3] U. Brenner,A. Pauli,J. Vygen, Almost Optimum Placement Legalization by Minimum Cost Flow and Dynamic Programming. ISPD 2004.

[4] U. Brenner and J. Vygen. Faster Optimal Single-Row Placement with Fixed Ordering. In Proc. Design, Automation and Test in Eurpoe, 2000.

[5] U. Brenner, and J. Vygen, Legalizing a placement with Minimum Total Movement, IEEE TCAD 23(12), pp 1597-1613. Dec 2004.

[6] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang. NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints. IEEE TCAD Integrated Circuits and Systems, 27(7):1228-1240, Jul 2008.

[7] M. Cho, H. Ren, H. Xiang, and R. Puri, History-based VLSI Legalization using Network Flow, pp 286-291, DAC10.

[8] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction to Algorithms, the MIT press.

[9] S. Dobre, A. Kahng, and J. Li, Mixed Cell-Height Implementation for Improved Design Quality in Advanced Nodes, pp 854-860, ICCAD15.

[10] K. Doll, F. M. Johannes, and K. J. Antreich, Iterative Placement Improvement by Network Flow Methods, pp 1189-1200, IEEE TCAD of Integrated Circuits and Systems, VOL13, No. 10, Oct 1994.

[11] A. Kahng, P. Tucker, A. Zelikovsky. Optimization of Linear Placements for Wirelength Minimization with Free Sites. ASPDAC, 1999.

[12] C. Kodama, H. Ichikawa, K. Nakayama, T. Kotani, Self-Aligned Double and Quadruple Patterning-Aware Grid Routing with Hotspots Control, ASPDAC, 2013.

[13] H. Ren, D. Z. Pan, C. J. Alpert, and P. Villarrubia. Diffusion-based Placement Migration. In Proc. Design Automation Conf., 2005.

[14] P. Spindler, U. Schlichtmann,F. M. Johannes. Abacus: Fast Legalization of Standard Cell Circuits with Minimal Movement. ISPD2008.