Resource-Aware Functional ECO Patch Generation

An-Che Cheng[1], Iris Hui-Ru Jiang[1] and Jing-Yang Jou[12]
[1]Department of Electronics Engineering, National Chiao Tung University, Hsinchu 30010, Taiwan
[2]Department of Electrical Engineering, National Central University, Taoyuan 32001, Taiwan

***Abstract*—Functional Engineering Change Order (ECO) is necessary for logic rectification at late design stages. Existing works mainly focus on identifying a minimal logic difference between the original netlist and the revised netlist, which is called a patch. The patch is then implemented by technology mapping using spare cells. However, there may be insufficient spare cells around the physical location of the patch, or the wires connecting spare cells are too long, thus causing timing violations and routing congestion. In this paper, we propose a resource-aware functional patch generation approach by gate count and wiring cost estimations. In particular, we estimate the number of spare cells required by a patch and define a cost of wire length on it, which considers the physical location of the patch and a set of nearby spare cells. As a result, the patch with minimal wiring cost instead of minimal size is produced. The experiments are conducted on nine industrial testcases. These testcases reflect real problems faced by designers, and the results show our method is promising.**

I. Introduction

Due to the increasing complexity of modern IC design, logic rectification or performance improvement are often required even at late design stages. Considering the time-to-market pressure and cost, it is impractical to restart the whole design flow from scratch. Instead, incremental design changes are made to achieve fixing with minimum resources. A widely used solution is Engineering Change Order (ECO). ECO can be performed at different design stages; the later stage where ECO is performed, the fewer changes are affordable and the greater challenges could be faced. In this paper, we focus on metal-only ECO, where ECO is performed after placement is frozen, and thus it is done by modifying only metal layers.

To facilitate metal-only ECO, spare cells are spread over a design during placement. These spare cells are redundant logical gates that do not affect the design functionality, and they also are not connected to any circuit components. When ECO is needed, the inputs and outputs of spare cells are rewired to the original circuit.

ECOs can be classified into two types by their objectives: functional ECO and timing ECO. Functional ECO, the focus of this paper, targets to fix bugs or revise specification, while timing ECO concentrates on repairing timing violations or improving circuit performance. In the literature, several automated functional ECO patch generation algorithms exist. State-of-the-art approaches can be mainly classified into two categories. One is the *error-detection-and-correction-based* approach [1]–[5]. This approach first borrows techniques from diagnosis and verification to identify possible rectifiable locations, and then derives *patches* (or called *rectification functions*) to fix the locations. The other is called the matching-based approach [6]–[8]. This method struggles to find structural matchings between the original and new circuits, then the logic differences can be extracted as patches.

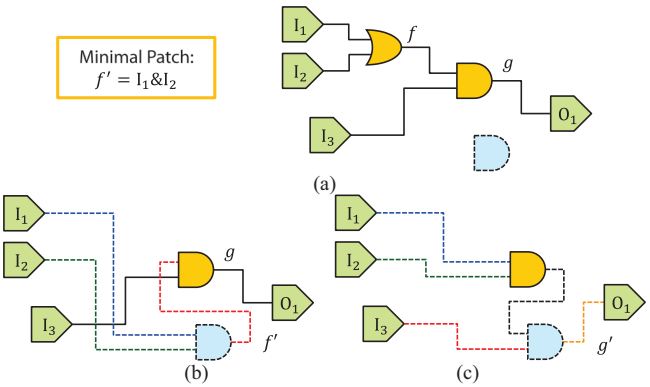


Fig. 1. Functional ECO considering physical location of spare cells. (a) a design with three inputs, one output, two logic cells and five nets. The given patch is $f' = I_1\&I_2$. (b) The netlist after implementing the patch. (c) The netlist after implementing the patch $g' = I_1\&I_2\&I_3$. The original AND2 logic cell is reused as a spare cell.

After ECO patches are generated, spare cells are then selected and rewired to implement them. Conventionally, existing methods consider these two steps independently. More specifically, in the patch generation step, previous approaches assume that finding a minimal patch can minimize the cost of the change. However, in real practice, there may be insufficient spare cells to implement the patch, or the wires connecting available spare cells are too long, thus causing timing violations. In either condition, the cost to make the change is not minimized, or even worse, the change cannot be completed. Obviously, the physical locations of the inputs and outputs of a patch and spare cells have a strong correlation to the final implemented netlist. Fig. 1 shows a motivational example. Fig. 1(a) presents a design with three inputs, one output, two logic cells and five nets. There is only one spare cell of type AND2. The given minimal patch indicates that net f should be modified to $I_1\&I_2$. The corresponding netlist after implementing the patch is shown in Fig. 1(b). Fig. 1(c) illustrates the netlist after implementing another patch $g' = I_1\&I_2\&I_3$. The original AND2 logic cell is reused as a spare cell. It can be seen that although g' is not the minimal patch, the wire length and routing congestion are better in the final netlist.

Therefore, in this paper, we propose a *resource-aware functional ECO patch generation* algorithm. Unlike prior works, which consider only the size of a patch, our approach takes the physical information of spare cells into consideration and thus can result in shorter wire length in the final netlist. For a primary output (PO) subset, our algorithm iteratively generate different patches that can fix it. During the iterations, the wire length of a patch is estimated by a cost function, and the patch with minimal cost instead of minimal size is kept. The cost function considers

978-3-9815370-7-9/DATE16/ ©2016 EDAA

two kinds of physical information: the average distance between the nearby spare cells, and the average distance from each spare cell to its ideal *virtual placement* location. Because the final patched netlist is unknown before technology mapping, the real number of required spare cells and the wire connections are also unknown. In this context, we use two different heuristic methods to estimate the gate count of a patch, and then perform virtual placement for these virtual gates. Finally, before the patch is outputted, a resubstitution procedure [9] is applied. In this step, we seek for opportunities to further minimize the patch by changing its supports because the patch size is directly related to the number of spare cells used. The experiments are conducted on nine industrial testcases that reflect real difficulties faced by designers. Experimental results show our method is promising.

The remainder of this paper is organized as follows. Section II gives some preliminaries and formulates the problem. Section III details our algorithm. Section IV demonstrates experimental results. Finally, Section V concludes this work.

II. Preliminaries and Problem Formulation

In this paper, the primary inputs (PIs) of a circuit are denoted as a set of Boolean variables $X = \{x_1, \dots, x_n\}$. The functions of the primary outputs in the original and new circuits are denoted by $F(X) = \langle f_1(X), \dots, f_m(X)\rangle$ and $G(X) = \langle g_1(X), \dots, g_m(X)\rangle$, respectively.

A. Rectification Functions and Interpolation

Definition 1. *Given original and new functions $F(X)$ and $G(X)$, $F(X)$ is said to be **single-rectifiable** with respect to $G(X)$ if there exists a Boolean function c, called the **single-fix rectification function**, such that $F(X, r = c(X)) \equiv G(X)$, where r is the **rectification signal** in F.*

Definition 2. *Given original and new functions $F(X)$ and $G(X)$, $F(X)$ is said to be **multi-rectifiable** with respect to $G(X)$ if there exists a set of Boolean functions $\{c_1, \dots, c_k\}$, called the **multi-fix rectification functions**, such that $F(X, r_1 = c_1(X), \dots, r_k = c_k(X)) \equiv G(X)$, where r_k's are the **rectification signals** in F.*

From the above definitions, the single-fix rectification function is a special case of the multi-fix rectification functions when $k = 1$. The necessary and sufficient conditions for the existence of the single-fix rectification function and multi-fix rectification functions were given in [1] and [5] respectively.

After a rectification signal r is determined, we can construct a rectification function by *interpolation*. The corresponding theorem is as below.

Theorem 1 (Craig Interpolation Theorem). [10] *Given two Boolean formulae A and B, with $A \wedge B$ unsatisfiable, there exists a Boolean formula I such that 1) $A \Rightarrow I$, 2) $I \wedge B$ is unsatisfiable, and 3) I refers only to the common variables of A and B.*

The Boolean formula I is called the *interpolant* of A and B. Given a refutation proof of A and B, the construction of an interpolant can be derived in linear time.

In the literature, several approaches construct patches based on this theorem [2]–[5]. In this paper, we adopt (but we are not limited to) the technique in [4] to construct a ***strong partial-fix rectification function***, such that no error minterm will be newly created and all correctable minterms for at least one PO will be rectified after applying the function [11].

B. Functional Dependency

Definition 3. *Given a Boolean function $f: B^m \to B$ and a vector of Boolean functions $E = \langle e_1(X), \dots, e_n(X)\rangle$ with $e_i: B^m \to B$ for $i = 1, \dots, n$, over the same set of variable vectors $X = \{x_1, \dots, x_m\}$, we say that f **functionally depends** on E if there exists a Boolean function $h: B^n \to B$ such that $f(X) = h(e_1(X), \dots, e_n(X))$. We call functions f, E, and h the **target function**, **base functions**, and **dependency function**, respectively.*

Note that in the definition, f and E are in the same domain, and h needs not depend on all of the functions e_i.

There are several applications of functional dependency in the field of electronic design automation (EDA). For example, it is useful in resubstitution and optimization of Boolean functions [9], and even in the functional ECO [2]. In our approaches, we apply the resubstitution technique in [9] to further simplify a patch.

C. Gate Count Estimation

Definition 4. [12] *Given a Boolean function F, the **occupancy** of F, $\phi(F)$, is defined as the fraction of minterms covered by the function.*

For example, function $F_1(a, b, c) = a + bc$ covers five out of the eight minterms. Therefore, the occupancy of F_1 is $5/8 = 0.625$.

Definition 5. [12] *Given a Boolean function $F(x_1, \dots, x_n)$, the Boolean difference of F w.r.t. i of its input variables ($1 \le i \le n$) is called the **i-fold derivative** of F. Let $S_i(F)$ represent the set of all the i-fold derivatives of F, the sum of occupancies of all the elements in $S_i(F)$ is defined as $\Phi_i(F) = \sum_{f \in S_i(F)} \phi(f)$, and the **area complexity metric**, $\mathcal{AC}(F)$, is defined as $\mathcal{AC}(F) = \sum_{i=1}^{n}(w_i \times \Phi_i)$, where w_i are weights assigned to Φ_i.*

For example, the 1-fold derivatives of F_1 are $S_1(F_1) = \left\{\frac{\partial F_1}{\partial a}, \frac{\partial F_1}{\partial b}, \frac{\partial F_1}{\partial c}\right\}$, where $\frac{\partial F_1}{\partial a} = F_{1_a} \oplus F_{1_{\bar{a}}} = \bar{b} + \bar{c}$ is the Boolean difference of F_1 w.r.t a, and $\Phi_1 = \sum_{f \in S_1(F_1)} \phi(f) = 3/4 + 1/4 + 1/4 = 5/4$; $\mathcal{AC}(F) = \sum_{i=1}^{3}(w_i \times \Phi_i)$.

Gate count or *area complexity* is a measure of the design complexity. An accurate estimate of gate count in the early stage of design development could be helpful to several aspects, such as power consumption prediction and evaluation of the influence of design change. A detail comparison of previous works related to gate count estimation is summarized in [13]. In this paper, we use a state of the art method [12] to predict the gate count of a patch without performing technology mapping.

D. Problem Formulation

The *resource-aware functional ECO* problem: Given a placed original circuit, including a set of placed spare cells, and a newly synthesized circuit, the goal is to generate patches such that wire length measured after technology mapping is minimized.

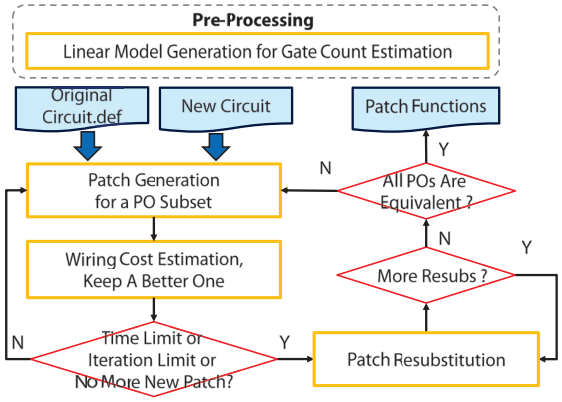


Fig. 2. The overall flow of our resource-aware functional ECO patch generation algorithm.

III. Resource-Aware Functional ECO Patch Generation

A. Algorithm Overview

The overall flow of the proposed algorithm is illustrated in Fig. 2. In the pre-processing step, a set of random circuits and a given library of spare cells are used to derive the linear regression coefficients for gate count estimation. Given a pre-placed and routed original circuit, which also contains a set of placed spare cells, and a newly synthesized circuit, a strong partial-fix rectification function is derived first to fix a subset of erroneous POs. Then, a cost related to the wire length of the patch is calculated. The algorithm for wiring cost estimation is shown in Fig. 3. Details of each step will be described later in the following sub-sections. Note that in this flow, we do not perform technology mapping, so the cost is only an approximation that measures the distance between the patch and a set of spare cells. The procedure then proceeds iteratively to find another strong partial-fix rectification function that can also fix the same PO subset. If it succeeds, the cost of the patches are compared, and the one with a lower cost is kept. The iteration stops when the time limit is reached or a given number of iterations is performed, or no more such patches exist. After that, a new iteration starts to resubstitute the patch. The objective of this phase is to further simplify the patch to reduce the required number of spare cells. When there is no possible resubstitution, the final patch is produced.

As it can be seen, our algorithm is flexible in many aspects. In the patch generation phase, different methods could be applied. Using different methods in parallel may also be effective. A patch can also be optimized by other techniques, while the reason we apply resubstitution is that it can re-perform a patch with different supports.

B. Pre-processing

In this step, we build a multiple linear regression model for gate count estimation. Following the flow of [12], we generate a set of ten thousand random PLA circuits [14]; 80% of all the circuits are used as training data to build the model, and the remaining 20% are for model validation. The derived coefficients of the model are the weights w_i in Definition 5. As a result, given a patch function F, we can estimate its gate count after technology mapping as $\mathcal{AC}(F) = \sum_{i=1}^{n}(w_i \times \Phi_i)$. In the experiments of [12], larger circuits are divided into single-output-ten-input circuits, and a single linear regression model is used. However, for patch circuits with fewer number of inputs, the estimation by a single model may have large deviation. Therefore, we derive the models from three-input to ten-input circuits.

C. Gate Count Estimation Approaches

Since the gate count predicted by area complexity does not guarantee to be accurate, we also use another prediction approach that calculates the number of AND2 nodes in the subject graph of a patch. We convert a patch into a subject graph that only has node types of PI, PO, AND2, and INV. Then the number of AND2 nodes in the subject graph is referred to as the *core size* of the patch in this paper, and is served as the estimated gate count. In the experiments, we compare the results produced by these two approaches.

D. Spare Cell Searching

Given the estimated gate count, e, we collect a set of $S = \rho \cdot e$ nearest spare cells of a patch, where ρ is a user-defined parameter not smaller than one. We use a 2-D range tree [15] to find spare cells in a bounding box in logarithmic time. First, the perimeter of the bounding rectangle that encloses the inputs and outputs of the patch is calculated. Then, a searching is performed at the bounding square of position centered at the mass center of the inputs and outputs of the patch, and with length equal to half of the computed perimeter. If the searching fails to collect enough spare cells, the area of the square is enlarged twice iteratively until S spare cells are collected or the whole design region is explored. During the search, the released logic gates are included. It was reported in [16] that using a range tree with quad-partition may speed up spare cell search about 1.49 times and 3.68 times compared to using a single-range tree and using a linear search, respectively.

E. Virtual Placement Approaches

In order to estimate the wire length more accurately, a quick virtual placement for a patch is performed to compute a relatively best location for each gate. Different approaches are used for the gates estimated by $\mathcal{AC}$ and core size.

1) Gates estimated from area complexity. Since the connections between the gates are unknown, we distribute the

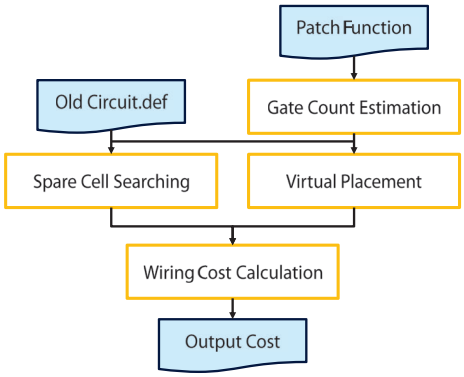
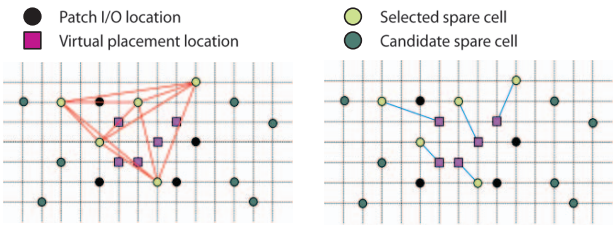
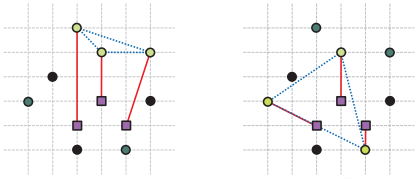


Fig. 3. The flow of our wiring cost estimation algorithm.



(a) (b)

Fig. 4. (a) Distances between each pair of a set of five spare cells. The four black nodes are the inputs and outputs of the patch. The five rectangles are the virtual placement locations. Other nodes are spare cells, and the light-colored ones are selected in current mapping. (b) Distances from a set of five spare cells to their ideal mapping locations.



(a) (b)

Fig. 5. (a) The virtual mapping with smallest D_s. The three black nodes are the inputs and outputs of the patch. The three rectangles are the virtual placement locations. Other five nodes are spare cells, and the light-colored ones are selected in current mapping. The solid lines indicate the distance from spare cells to their mapped virtual locations, while the dashed lines represent the distance between the spare cells. (b) The virtual mapping with smallest D_l.

gates evenly in the bounding rectangle of inputs and outputs of a patch by the method of ***Centroidal Voronoi Tessellations*** (CVT) [17]. A CVT is a special type of *Voronoi tessellation* in which the generating point in each of the Voronoi regions is also the centroid (center of mass) of it.

DEFINITION 6. [18] *Given a set* $T = \{t_1, \dots, t_n\}$ *of* n *points in a plane* I, *A* ***Voronoi diagram*** *(Voronoi tessellation) is a decomposition of* I *into a set of* n *disjoint Voronoi regions that satisfy:*

$$D(x, t_i) \leq D(x, t_j),\ x \in I,\ \forall t_{i,j} \in T, \mathrm{i} \neq \mathrm{j} \qquad (1)$$

The points are referred to as the *generators* of the diagram. Initially, we roughly place the gates in the bounding box of a patch. Then, we apply CVT to evenly distribute the gates.

2) Gates estimated from subject graph. We use the greedy heuristic method proposed in [19] to compute the ideal locations of the nodes in the subject graph of a patch. A little difference from it is that we omit the inverters. In each path from PI to PO in the subject graph, the candidate locations of the AND2 nodes are calculated by averaging the distance of the nodes along the path. If an AND2 node appears in more than one path, its ideal location is given by the mean of the candidate locations.

F. Wiring Cost Calculation

After the virtual placement locations are determined, the wiring cost of a patch is calculated.

DEFINITION 7. *Given a set of* e *spare cells, and a set of* e *virtual placement locations, a* ***virtual mapping*** *is a mapping from these spare cells to the virtual placement locations with minimal total distance from each spare cell to its ideal location*.

We define the ***wiring cost*** of a virtual mapping as below:

$$W_m = D_s + D_l. \qquad (2)$$

where the notation D_s is defined as the average distance between the spare cells, and D_l is defined as the average distance from each spare cell to its ideal virtual placement location. An illustrative example is shown in Fig. 4. There are five chosen spare cells (light-colored), and the line segments in Fig. 4(a) indicate the distances between them, so D_s is the average of these distances. Similarly, the line segments in Fig. 4(b) represent the distances from the spare cells to their ideal locations, and D_l is the average of these distances. Essentially, D_s measures how close are the spare cells, and D_l measures how far are the spare cells from the ideal locations. Obviously, these two factors have strong relationships with the final wire length after technology mapping.

Instead of using the cost of the virtual mapping by the nearest e spare cells of the patch, we seek for a representative cost from all the $\binom{S}{e}$ virtual mappings. The ***representative wiring cost*** is defined as below:

$$W_r = \frac{1}{2}\left(W_m\,(min.D_s) + W_m\,(min.D_l)\right), \qquad (3)$$

where $W_m\,(min.D_s)$ is the wiring cost of the virtual mapping with minimal D_s, and $W_m\,(min.D_l)$ is the wiring cost of the virtual mapping with minimal D_l. For example, a patch with $e = 3$ and $S = 5$ is shown in Fig. 5. The mappings with the smallest D_s and smallest D_l are illustrated in Fig. 5(a) and Fig. 5(b) respectively. The wiring cost of the corresponding mapping in Fig. 5(a) is $W_m = D_s + D_l = 2.19 + 3.05 = 5.24$. Similarly, $W_m = D_s + D_l = 4.07 + 1.75 = 5.82$ in Fig. 5(b). By the formula (3), the representative cost W_r of the patch is $(5.24 + 5.82)/2 = 5.53$.

To derive the mapping with minimal D_s, first all pairwise distances of spare cells are calculated and recorded in a lookup table, then D_s of each virtual mapping is iteratively computed by adding the pre-calculated distances, and the mapping with minimal D_s is kept. On the other hand, finding the mapping with minimal D_l is more complicated. We model it as the *minimum-cost bipartite matching* problem and solve it by the Hungarian algorithm [20]. The two independent sets in the bipartite graph are the set of spare cells and the set of virtual locations, and a cost of an edge is the distance between a spare cell and a virtual placement location. However, the Hungarian algorithm could be best implemented to be running in time complexity $O(n^3)$. Empirically, when the number of spare cells exceeds 40, the computing time would consume several minutes. Hence, for larger patches, we reduce the mapping problem to the *stable marriage problem* [21]:

The Stable Marriage Problem: Given a set of men and women, each man has ranked the women in order of preference and each woman has done likewise, marry them off in pairs such that no pair of man and woman would both prefer each other to their current partners. If there are no such pairs, all the marriages are stable.

In our modeling, the virtual locations represent the men, and the spare cells represent the women. Consequently, the male-optimal Gale–Shapley algorithm [21] can derive a stable matching for the locations and the spare cells in time complexity $O(n^2)$.

G. Patch Resubstitution

When a patch with minimal wiring cost is derived, we perform Boolean resubstitution to further simplify it. Note that in the metal-only ECO flow, the number of pre-placed spare cells is fixed. Nevertheless, ECOs are usually performed not only once, so saving more spare cells is implicitly facilitating the producing of minimal patches afterwards.

The method of [9] used in our flow is briefly described in this subsection. The node to be optimized (in this case, output of a patch) is called the *pivot*, and a local neighborhood of the pivot is called a *window*. Then the transitive fanin (TFI) cone of the pivot is computed with at most *tfi_level_max* levels, and the PIs of the window are also computed. Next, the transitive fanout (TFO) cone of the pivot is computed with at most *tfo_level_max* levels, and the POs of the window are computed. The paths between the PIs and the POs of the window are marked, while those going through the pivot node are skipped. The nodes in the TFO cone without marked paths are removed and the POs of the reduced TFO cone are re-computed. Finally, the nodes on the path from the PIs to the POs of the window are the *base functions* for the patch as the *target function*. Given a target function and a set of base functions, whether a dependency function exists can be solved by incremental satisfiability (SAT) solving and interpolant computation [22]. A simple example is as below: suppose we have a patch $f = (a \oplus b)(b + c)$, and there are two base functions: $e_1 = a'b$ and $e_2 = ab'c$, a dependency function $h = e_1 + e_2$ could be found for f.

IV. Experimental Results

A. Experimental Settings

Our algorithm was implemented in C language integrated with ABC [23]. All experiments are conducted on a Linux machine with Intel Xeon CPU E5-2630 of 2.30 GHz frequency and 64 GB memory.

Table I shows the regression results of area complexity of circuits with different number of inputs. For the circuits with input number n smaller than five, we calculate Φ_i up to Φ_n; otherwise, we calculate Φ_i up to Φ_5. A good regression model should have high R^2(up to 1) and high correlation value (up to 1), but low error rate. From Table I, we can see that when the number of inputs is lower than four, the error rate increases drastically.

There are eight industrial testcases used in the experiments. The library of spare cells for testcases 1–3 contains basic logic cells and multiplexors, while testcase4 and testcase5 use only multiplexors and inverters. The statistics of testcases is listed in Table II, including the number of pins (#Pins), the number of cells (#Cells), the number of nets (#Nets), and the number of spare cells (#Spare). For testcases 2 and 3, spare cells are mainly located at corners; in the testcases 1*–3*, the circuits are the same as testcases 1–3 but the number of spare cells is largely reduced. These testcases reflect the real difficulties faced by designers. All the netlists and placements of original designs are described in DEF format [24]. The parameter for ρ and the iteration limit are set to 1.1 and 20, respectively. We use the **ECOS** metal-only ECO synthesizer [25] to perform technology mapping after functional ECO patches are generated.

Table I
Regression Results of Area Complexity

#PIs	R^2	COR	%ERROR
3	0.681	0.818	63.019
4	0.720	0.891	35.210
5	0.953	0.841	6.037
6	0.879	0.890	8.196
7	0.873	0.920	11.787
8	0.926	0.967	2.109
9	0.942	0.966	1.956
10	0.943	0.969	2.181

Table II
Testcase Statistics

Case	# Pins	# Cells	#Nets	#Spare
testcase1	483	28,591	28,705	350
testcase2	483	28,591	28,705	2,300
testcase3	483	28,591	28,705	2,300
testcase4	33	198	181	40
testcase5	30	938	850	100
testcase1*	483	28,591	28,705	35
testcase2*	483	28,591	28,705	50
testcase3*	483	28,591	28,705	50

B. Resource-Aware Functional ECO Results

Table III summarizes experimental results of our algorithm. We compare our results with those of the minimal patches. Column 2 shows the type of changes made on the original circuits. We perform two types of modifications on the original netlists by changing the logic operations (logic), and changing the wire connections (wire). Columns 3 and 4 are the patch sizes and the total wiring costs after technology mapping of minimal patches. The corresponding patch size, total wiring cost, and runtime of our methods are listed in Column 5, Column 6, and Columns 8-9, respectively. The runtime of minimal patch generation are not shown because they are recorded during the patch generating iterations of our algorithm. The notation $\mathcal{AC}$ means we use area complexity to estimate the gate count and do virtual placement by CVT. On the other hand, we also predicate the gate count by the number of AND2 nodes in a subject graph, and do virtual placement by the greedy heuristic.

In the results of testcase1, the minimal patch is also the one with minimal wiring cost. Empirically, this situation occurs when spare cells are sufficient and evenly distributed. In other testcases, the minimal patches are different from the patches with minimal wiring cost. For testcase1*, the minimal patch is the same as testcase1, but the minimal patch does not lead to minimal wiring cost anymore. For testcase2 and testcase2*, the corresponding patches are the same, but wiring cost increases when the number of spare cells is fewer. Similar results are also shown in testcase3 and testcase3*. The results in Table III show that the patches generated by our method can be implemented using shorter wire length, achieving 28.08% reduction on average.

Both the two different estimation approaches can produce patches with lower wiring cost. However, we can see that the runtimes of $\mathcal{AC}$ are slightly longer. There are two reasons: 1)

TABLE III
EXPERIMENTAL RESULTS

Case	Error Type	Minimal Patch		Resource-Aware Patch			Total Time (s)	
		Patch Size (#AIG node)	Resulting Wiring Cost (HPWL) (A)	Patch Size (#AIG node)	Resulting Wiring Cost (HPWL) (B)	Reduction ((A-B)/A)	$\mathcal{AC}$	Subject Graph
testcase1	3 logic	11	1,957,960	11	1,957,960	0.00%	110.13	87.6
testcase2	2 logic	2	3,357,580	6	1,953,100	41.83%	1920.56	1910.74
testcase3	1 logic + 1 rewire	3	93,536,850	4	85,784,280	8.29%	799.95	795.26
testcase4	2 logic	3	40,320	5	28,280	29.86%	0.27	0.09
testcase5	1 logic + 1 rewire	3	223,790	3	164,990	26.27%	16.53	16.47
testcase1*	3 logic	11	7,082,980	12	4,581,080	35.32%	38.52	16.41
testcase2*	2 logic	2	4,422,480	6	2,550,380	42.33%	70.27	69.33
testcase3*	1 logic + 1 rewire	3	103,965,860	4	90,794,360	12.67%	795.49	791.18
Avg.		5		6		28.08%		
Ratio							1.02	1.00

for smaller patches, the estimated gate counts are usually larger than the real gate counts, so it takes more time on computing virtual locations; 2) for larger patches, the computing times of CVT are longer than those of greedy heuristic. We also observed that it is hard to find a patch which has smaller wiring cost but is much larger than the minimal patch. Finally, the reason of the longest runtime in testcase2 is that the patch is located at a resource-intensive place, so our algorithm takes much time on filtering the spare cells for virtual placement during the iterations.

V. CONCLUSIONS

Existing methods for functional ECO patch generation did not consider the physical information of spare cells. In contrast, in this paper, we have proposed a novel approach that considers the available resources, i.e. spare cells placed around a patch. By estimating the gate count of a patch from its subject graph or area complexity and performing virtual placement for this patch, we can estimate the wiring cost after technology mapping. We have devised a cost function that reflects the average distance between a set of spare cells and the average distance from the spare cells to the virtual placement locations. Experimental results have demonstrated that our algorithm is effective in producing patches with shorter wire length.

REFERENCES

[1] C.-C. Lin, K.-C. Chen, and M. Marek-Sadowska, "Logic synthesis for engineering change," *IEEE Trans. Comput. Design Integr. Circuits Syst.*, vol. 18, no. 3, pp. 282–292, 1999.

[2] A. C. Ling, S. D. Brown, S. Safarpour, and J. Zhu, "Toward automated ECOs in FPGAs," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 30, no. 1, pp. 18–30, 2011.

[3] B.-H. Wu, C.-J. Yang, C.-Y. Huang, and J.-H. R. Jiang, "A robust functional ECO engine by SAT proof minimization and interpolation techniques," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, pp. 729–734, 2010.

[4] K.-F. Tang, C.-A. Wu, P.-K. Huang, and C.-Y. Huang, "Interpolation-based incremental ECO synthesis for multi-error logic rectification," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, pp. 146–151, 2011.

[5] K. Tang, P. Huang, C. Chou, and C. R. Huang, "Multi-patch generation for multi-error logic rectification by interpolation with cofactor reduction," in *Proc. Design Autom. Test in Europe Conf. Exhib. (DATE)*, pp. 1567–1572, 2012.

[6] S. Krishnaswamy, H. R. H. Ren, N. Modi, and R. Puri, "DeltaSyn: An efficient logic difference optimizer for ECO synthesis," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)- Dig. Tech. Paper*, 2009.

[7] H. Ren, R. Puri, L. Reddy, S. Krishnaswamy, C. Washburn, J. Earl, and J. Keinert, "Intuitive ECO Synthesis for High Performance Circuits," in *Proc. Design Autom. Test in Europe Conf. Exhib. (DATE)*, pp. 1002–1007, 2013.

[8] S.-L. Huang, W.-H. Lin, P.-K. Huang, and C.-Y. Huang, "Match and replace: A functional ECO engine for multierror circuit rectification," *IEEE Trans. Comput. Design Integr. Circuits Syst.*, vol. 32, no. 3, pp. 467–478, 2013.

[9] A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang, "Scalable don't-care-based logic optimization and resynthesis," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 4, pp. 1–23, 2011.

[10] W. Craig., "Linear reasoning: A new form of the Herbrand-Gentzen theorem," *J. Symb. Log.*, vol. 22, no. 3, pp. 250–268, 1957.

[11] S. Y. Huang, K. C. Chen, and K. T. Cheng, "AutoFix: A hybrid tool for automatic logic rectification," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 18, no. 9, pp. 1376–1384, 1999.

[12] A. Kagliwal and S. Balachandran, "Measuring area-complexity using Boolean difference," in *Proc. IEEE Int. Conf. VLSI Design*, pp. 245–250, 2013.

[13] V. Eerola and J. Nurmi, "High-level parameterizable area estimation modeling for ASIC designs," *Integr. VLSI J.*, vol. 47, no. 4, pp. 461–475, 2014.

[14] T. Měchura & P. Fišer, (c) 2008, Random Circuits Generators. [Online] Available: http://ddd.fit.cvut.cz/prj/Circ_Gen/index.php?page=pla.

[15] S. S. Dinesh P. Mehta, Dinesh P. Mehta, *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004.

[16] K.-H. Ho, J.-H. R. Jiang, and Y.-W. Chang, "TRECO: Dynamic technology remapping for timing engineering change orders," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 31, no. 11, pp. 1723–1733, 2012.

[17] Q. Du, V. Faber, and M. Gunzburger, "Centroidal Voronoi Tessellations: Applications and Algorithms," *SIAM Rev.*, vol. 41, no. 4, pp. 637–676, 1999.

[18] G. Voronoi, "Nouvelles applications des parametres continus a la theorie des formes quadratiques.," *J. fur die reine und Angew. Math.*, vol. 134, pp. 198–287, 1908.

[19] K.-H. Ho, Y.-P. Chen, J.-W. Fang, and Y.-W. Chang, "ECO timing optimization using spare cells and technology remapping," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 29, no. 5, pp. 697–710, 2010.

[20] H. W. Kuhn, "The Hungarian method for the assignment problem," *Nav. Res. Logist. Q.*, vol. 2, no. 1–2, pp. 83–97, 1955.

[21] D. G. and L. S. Shapley, "College admissions and the stability of marriage," *Amer. Math. Mon.*, vol. 69, pp. 9–14, 1962.

[22] J.-H. R. Jiang, C.-C. Lee, A. Mishchenko, and C.-Y. Huang, "To SAT or Not to SAT: Scalable Exploration of Functional Dependency," *IEEE Trans. Comput.*, vol. 59, no. 4, pp. 457–467, 2010.

[23] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification. http://www.eecs.berkeley.edu/~alanmi/abc/

[24] LEF/DEF Exchange Format: Reference Documentation Plus Parser [Online]. Available: http://www.si2.org/openeda.si2.org/projects/lefdef.

[25] I. H.-R. Jiang and H.-Y. Chang, "ECOS: Stable matching based metal-only ECO synthesis," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 20, no. 3, pp. 485–497, 2012.