Cost-Aware Patch Generation for Multi-Target Function Rectification of Engineering Change Orders

He-Teng Zhang, and Jie-Hong R. Jiang
Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, 10617, Taiwan
{superpi152@gmail.com, jhjiang@ntu.edu.tw}

ABSTRACT

The increasing system complexity makes engineering change order (ECO) mostly inevitable and a common practice in integrated circuit design. Despite extensive research being made, prior methods are not effectively applicable to instances where rectification is to be done by simultaneously fixing multiple target points using intermediate signals. Moreover, how to efficiently generate low-cost patch functions is rarely addressed. These challenges are posed as a problem in the 2017 ICCAD CAD Contest. Based on Boolean satisfiability and interpolation, we propose a sound and complete algorithm for resource-aware patch generation of multi-target ECO. Experiments show our high quality results compared to other winning teams in the contest.

1. INTRODUCTION

The increasing complexity of system design makes engineering change order (ECO) an unavoidable process in integrated circuit design. ECO aims to rectify a given flawed design in a cost effective way by reusing prior design efforts as much as possible because rerunning the entire synthesis flow from scratch is largely impractical for time-to-market and other considerations. Moreover, we require the rectification solution to be physically feasible, that is, meeting physical design resource constraints.

There has been intensive work on ECO methods at different design stages. For example, ECO during logic synthesis was studied in [15, 14, 21, 19, 20]; ECO during physical synthesis was studied in [3, 7].

For functional ECO, logic synthesis for ECO was formulated in the early work [15]. The authors in [14] proposed a method combining structural and functional techniques to maximize reusable subcircuits to make ECO practical. More recently, SAT solving [6] and *interpolation* [16] techniques were exploited for scalable functional ECO computation [21, 19]. In [21], only single-target rectification was addressed. The extension to multi-error logic rectification was investigated in [19, 20]. The multi-error fix in [19] was done by incremental partial rectification of error minterms, and cannot guarantee successful ECO when the rectification targets are pre-specified. The problem was resolved in [20] assuming that the patch functions take primary inputs as their support inputs. However, how to generate patch functions in terms of intermediate signals, rather than primary inputs, remains challenging.

For physical ECO, design issues, such as timing and spare cell availability, have to be addressed [3, 7]. Recently, functional ECO patch generation considering physical resource issues was addressed in [4]. How to efficiently derive low-cost patch functions remains an unsolved problem and is the main challenge posed by the 2017 ICCAD CAD Contest (Problem A) [11]. In this work, we tackle the contest problem.

DAC '18, June 24–29, 2018, San Francisco, CA, USA
Copyright 2018 ACM 978-1-4503-5700-5/18/06
https://doi.org/10.1145/3195970.3196017 ...$15.00.

The computation of functional ECO can often be divided in two main phases: First, identify target signals for rectification. Second, given the identified target signals, derive cost effective patch functions. In this work, we focus on the latter task and propose 1) a sound and complete algorithm for multi-target ECO patch generation, 2) an effective heuristics for cost-aware patch input selection, and 3) an enhancement technique to make interpolation applicable to small patch synthesis using don't cares. Experimental results show that our method outperforms the top winning team of CAD Contest [9] in terms of both patch cost and patch size, especially on difficult ECO instances, while the runtime stays reasonable.

This paper is organized as follows. Section 2 defines essential terminologies. An overview of our computation flow is given in Section 3. Section 4 presents our main algorithm for multi-fix ECO. In Section 5, we enhance the applicability of interpolation for low-cost and small-size patch derivation. In Section 6, we explain the main ideas of our cost optimization algorithm. Section 7 evaluates our methods in comparison to the 1st place results of the 2017 CAD Contest. Finally Section 8 concludes this work.

2. PRELIMINARIES

2.1 Boolean Circuit

A *Boolean circuit* $G = (V, E)$ is a directed acyclic graph (DAG) consisting of vertices V and edges $E \subseteq V \times V$. An edge $e = (u, v)$ signifies that vertex v refers to vertex u as an input; u is called a *fanin* of v; v is called a *fanout* of u. A vertex in V without fanins (resp. fanouts) is a *primary input* (resp. *primary output*) of the circuit. A vertex v except for the primary inputs represents a logic gate. Every vertex is associated with a Boolean variable, called a *signal*, whose valuation is arbitrary for a primary input or is determined by the corresponding logic gate in terms of its fanins.

2.2 Engineering Change Order

In the (functional) *engineering change order* (ECO) problem, we are given two circuits F and G, which are referred to as the *faulty circuit* and *golden circuit*, respectively, and are asked to change F with modification minimum in some cost measure such that F after the modification is functionally equivalent to G. In the sequel, we assume that the primary outputs of F and G are of functions $F(X) = (f_1(X), \ldots, f_m(X))$ and $G(X) = (g_1(X), \ldots, g_m(X))$, respectively, specified in terms of primary input variables $X = (x_1, \ldots, x_n)$.

If F and G are functionally non-equivalent, then there exists some assignment x^* to X, called an *error minterm*, such that $f_j(x^*) \neq g_j(x^*)$ for some j, that is, $\exists X.F(X) \neq G(X)$. To make F functionally equivalent to G, a set of *target signals*

$T = (t_1, ..., t_\alpha)$ in F is to be rectified. A *patch* is referred to as a set of *patch functions* $P(B) = (p_1(B_1), ..., p_\alpha(B_\alpha))\}$, where $B = \cup_i B_i$, called the *base* of the patch, is a set of signals in circuit F such that substituting patch function p_i to target signal t_i, for $i = 1, \ldots, \alpha$, makes F functionally equivalent to G. When $\alpha > 1$, the rectification problem is referred to as *multi-fix ECO.*

The *multi-fix ECO problem* considered in this work (also CAD Contest 2017) assumes that the target signals T are pre-specified. Specifically, the original faulty circuit $F(X) = (f_1(X), \ldots, f_m(X))$ has been rewritten as $F(X, T) = (f_1(X, T), \ldots, f_m(X, T))$ such that the target signals are floating (without fanins) and treated as the pseudo-PIs of F. Moreover, each signal in the circuit F is associated with a cost. The quality of a patch $P(B)$ in our (CAD Contest 2017) formulation is determined by two factors: the summation of the costs of the base signals and the circuit size that realizes the patch functions.

2.3 Care-set and Diff-set

In the context of ECO, for a function $f_j(X, t)$ with a single target input t, we define its *care-set* with respect to t as

$$care^t_j(X) = f_j|_{t=0} \oplus f_j|_{t=1},$$

where $f_j|_{t=e}$ denotes substituting variable t in f_j with e, which can be a constant or more generally a function. The care-set forms a set of assignments to X, under which the valuation of $f_j(X, t)$ is sensitive to the value of signal t. For a function $f_j(X, T)$ with target inputs T, we define its *care-set* with respect to $t_k \in T$ as

$$care^{t_k}_j(X, T \setminus \{t_k\}) = f_j|_{t_k=0} \oplus f_j|_{t_k=1}.$$

We define the *diff-set*, denoted $diff_j$, as the Boolean difference between the j^{th} output functions f_j and g_j of circuits F and G by

$$diff_j(X) = f_j(X) \oplus g_j(X).$$

In other words, a diff-set is a set of error minterms, by which we can distinguish $f_j(X)$ from $g_j(X)$. For a function $f_j(X, T)$ with target inputs T, we define the diff-set of f_j as

$$diff_j(X, T) = f_j(X, T) \oplus g_j(X).$$

In the sequel, for single-output circuits F and G, the subscripts j in the above expressions are omitted for brevity.

2.4 Functional Dependency and Don't Care Optimization

Given a function over primary input variables, it can be reexpressed as another function over a set of selected driving functions through the *functional dependency* formulation [12]. It can be exploited to search a good set of driving signals such that a function can be rewritten and optimized in terms of gate counts or delay. After a feasible set of driving signals is found through satisfiability solving [6], the function can be synthesized using *interpolation* [5], which is based on the following theorem.

THEOREM 1 (CRAIG INTERPOLATION THEOREM). *[5] Given two Boolean formulas ϕ_A and ϕ_B, with $\phi_A \wedge \phi_B$ unsatisfiable, then there exists a Boolean formula ψ_A referring only to the common variables of ϕ_A and ϕ_B such that $\phi_A \rightarrow \psi_A$ and $\psi_A \wedge \phi_B$ is unsatisfiable.*

Essentially, the interpolant can be construction from the refutation proof of the unsatisfiability of $\phi_A \wedge \phi_B$ returned by a SAT solver [16].

In the interpolation-based synthesis, circuit don't cares can be exploited for further optimization [17]. Don't care optimization is especially important in ECO because the observability don't cares (ODCs) of the target signals could be present to some extent and satisfiability don't cares (SDCs) could be helpful in exploring better driving signals.

3. ALGORITHMIC FLOW

Figure 1 gives an overview of our ECO flow. The computation takes the faulty and golden circuits as input. It first detects functionally equivalent vertices among the two circuits by functionally reduced and-inverter graph (FRAIG) computation [18]. Vertices of the same functionality are placed into the same equivalence class. We take advantage of these equivalence classes for two purposes: 1) to identify shared equivalent signals among faulty and golden circuits for our patch generation algorithm, and 2) to reduce the computation overhead in the downstream optimization stage.

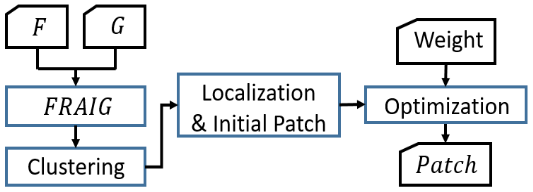

Figure 1: Overview of our ECO flow.

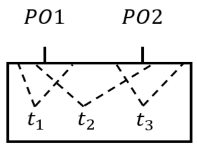

Figure 2: An example of clustering stage in our flow.

Next, a clustering operation is performed. In the clustering, two target signals in circuit F are placed in a group if they share some primary output in their transitive fanout cones. It iteratively merges two groups if they share some target signal. For example, in Figure 2, target signals t_1, t_2 and t_3 will be put in one group after the clustering process. The purpose of clustering is to reduce the computation overhead in later stages. By doing so, we can rectify target points one group at a time.

In the localization stage, we handle target points one group at a time. With the equivalence classes obtained from the FRAIG stage, we identify shared equivalent signals among faulty and golden circuits. Then we tag these shared equivalent signals in the golden circuit. These tagged signals will form a cut, by which we can further reduce the size of relevant subcircuit and the initial patch. Localization can also enhance the applicability of interploation-based synthesis [21, 19], as to be discussed in later sections.

After localization, an initial patch is generated by our patch generation algorithm. The optimization stage takes a *weight* file as input, which specifies the weight of each internal signal. It searches a patch with the lowest cost, where the cost is calculated by the sum of the weights of the base signals. This stage could iterate several rounds until no more improvements can be achieved.

4. MULTI-FIX ECO PATCH GENERATION

We first propose a multi-fix ECO algorithm for single-output circuits, and then extend it to multi-output circuits.

4.1 Rectifiability

Prior work [19, 20] partly addressed the multi-fix ECO problem. However both prior methods are not effectively applicable to the CAD Contest challenge. Specifically, prior work [19] is incomplete for multi-fix ECO. In [19], the on-set of a patch function p_k is written as

$$p_k^{on}(X) = \begin{array}{l} care^{t_k}(X) \wedge \mathit{diff}(X) \wedge \neg e_k(X) \vee \\ care^{t_k}(X) \wedge \neg \mathit{diff}(X) \wedge e_k(X), \end{array} \quad (1)$$

where $care^{t_k}(X)$ characterizes a set of minterns, under which the value change of signal t_k is observable at primary outputs, $\mathit{diff}(X)$ is a set of error minterms, under which F and G are non-equivalent, and $e_k(X)$ is the *erroneous function* in the original faulty circuit $F(X)$ that drives target signal t_k. In [19], the set of target signals are selected on-the-fly during iterations of partial fix. When the prior method [19] is applied in our multi-fix ECO formulation, where target signals are given *a priori*, it is possible that fixing an erroneous function $e_i(X)$ might make others $e_j(X)$ unrectifiable.

The above problem was resolved in [20]. For a given set of target signals T, the rectifiability of $F(X,T)$ to G can be expressed by

$$\forall X, \exists T. F(X,T) = G(X). \quad (2)$$

In essence, the faulty circuit F can be rectified through target signals T if and only if Eq. (2) holds. If the quantified Boolean formula of Eq. (2) is true, then the *Skolem certificates* [1] of variables T correspond to the desired patch functions. However, the patch functions refer only to the primary input signals X, rather than intermediate signals in circuit F. In this work, we take both primary input and intermediate signals into account for the generation of cost-effective patch functions.

4.2 Patch Generation for Single-Output Circuit

Observe that if the faulty circuit F is rectifiable with target signals T, then the primary outputs must be sensitive to T under the error minterms. For a single-output faulty circuit F, its error minterms that are potentially rectifiable by a target signal t_k can be characterized by

$$care^{t_k}(X, T \setminus \{t_k\}) \wedge \mathit{diff}(X,T). \quad (3)$$

Based on Eq. (3), similar to the Boolean relation determinization procedure in [13] we generate patch functions for target signals one by one in two phases as follows. In the first phase, we derive the target-variable dependent patch function $p'_k(X, t_{k+1}, \ldots, t_\alpha)$ of target t_k for k from 1 to α. To simplify the expression, we define T_k as

$$T_k = \begin{cases} T \setminus \{t_1\}, & \text{if } k = 1 \\ T_{k-1} \setminus \{t_k\}, & \text{if } k > 1 \end{cases} \quad (4)$$

and rewrite $p'_k(X, t_{k+1}, \ldots, t_\alpha)$ as $p'_k(X, T_k)$. Essentially, the on-set and off-set of p'_k can be characterized by

$${p'}_k^{on}(X,T_k) = (care^{t_k} \wedge \mathit{diff}|_{t_k=0})|_{t_1=p'_1,\ldots,t_{k-1}=p'_{k-1}} \quad (5)$$

$${p'}_k^{off}(X,T_k) = (care^{t_k} \wedge \mathit{diff}|_{t_k=1})|_{t_1=p'_1,\ldots,t_{k-1}=p'_{k-1}} \quad (6)$$

The target-variable dependent patch function $p'_k(X,T_k)$ can be obtained by computing the interpolant ψ_A from the unsatisfiability of $\phi_A \wedge \phi_B$ with $\phi_A = {p'}_k^{on}(X,T_k)$ and $\phi_B = {p'}_k^{off}(X,T_k)$ as stated in Theorem 1. Alternatively, one can simply take ${p'}_k^{on}(X,T_k)$ or $\neg {p'}_k^{off}(X,T_k)$ as an implementation for $p'_k(X,T_k)$.

The procedure of deriving $p'_k(X,T_k)$ is outlined in Algorithm 1. In line 7, procedure SynthesizePatch computes the patch function through interpolation. In line 8, the faulty circuit F' is updated by connecting the newly generated patch p'_k to target signal t_k. After the while-loop ends, the faulty circuit has no more target signals. In line 11, the generated target-variable dependent patch functions are returned.

Algorithm 1 DependentPatchGen

```
 1: procedure DependentPatchGen($F$,$T$,$G$)
 2:     $\alpha \leftarrow |T|$
 3:     $k \leftarrow 1$                         ▷ To rectify the $k^{\text{th}}$ target signal
 4:     $F' \leftarrow F(X,T)$                   ▷ A copy of faulty circuit
 5:     while $k \le \alpha$ do
 6:         $T_k \leftarrow \{t_{k+1}, ..., t_\alpha\}$
 7:         $p'_k(X,T_k) \leftarrow$ SynthesizePatch($F', T_k, G, k$)
 8:         $F' \leftarrow F'|_{t_k=p'_k}$              ▷ Update faulty circuit
 9:         $k \leftarrow k+1$
10:     end while
11:     return $\{p'_1(X,T_1), p'_2(X,T_2), ..., p'_\alpha(X)\}$
12: end procedure
```

In the second phase, we eliminate the dependency on the target variables from the generated $p'_1, \ldots, p'_\alpha$ by the following substitutions.

$$\begin{array}{ll} p_\alpha(X) & = p'_\alpha(X) \\ p_{\alpha-1}(X) & = p'_{\alpha-1}(X, t_\alpha = p_\alpha(X)) \\ p_{\alpha-2}(X) & = p'_{\alpha-2}(X, t_{\alpha-1} = p_{\alpha-1}(X), t_\alpha = p_\alpha(X)) \\ & \vdots \\ p_1(X) & = p'_1(X, t_2 = p_2(X), t_3 = p_3(X), \ldots, t_\alpha = p_\alpha(X)) \end{array}$$

The correctness of the above two-phase derivation is similar to that of the Boolean relation determinization procedure [13]. The proof can be done through induction on k and is omitted due to space limitation.

4.3 Patch Generation for Multi-Output Circuit

When F (or G) has m outputs, for $m \ge 1$, Eq. (5) and Eq. (6) can be extended as follows.

$${p'}_k^{on}(X,T_k) = \bigvee_{j=1}^{m} (care_j^{t_k} \wedge \mathit{diff}_j|_{t_k=0})|_{t_1=p'_1,\ldots,t_{k-1}=p'_{k-1}} \quad (7)$$

$${p'}_k^{off}(X,T_k) = \bigvee_{j=1}^{m} (care_j^{t_k} \wedge \mathit{diff}_j|_{t_k=1})|_{t_1=p'_1,\ldots,t_{k-1}=p'_{k-1}} \quad (8)$$

Algorithm 1 can be directly applied to generate the target-variable dependent patch functions for further derivation of target-variable independent patch functions. The only modification needed is in procedure SynthesizePatch relying on Eq. (7) and Eq. (8), rather than Eq. (5) and Eq. (6).

However, note that applying interpolation over Eq. (7) and Eq. (8) in SynthesizePatch to synthesize the patch function may fail. To understand the cause of the potential failure, assume that we use interpolation-based synthesis, and the faulty circuit F is rectifiable by target signals $T = (t_1, \ldots, t_\alpha)$. The patch function $p'_k(X,T_k)$ could be synthesized from the *refutation proof* of the formula

$${p'}_k^{on}(X,T_k) \wedge {p'}_k^{off}(X,T_k), \quad (9)$$

only if the formula is unsatisfiable. However, there may exist a truth assignment on variable (X,T_k), say x^* and t^*, under which the following formula holds

$$\begin{array}{c} \left(care_i^{t_k} \wedge \mathit{diff}_i|_{t_k=0}\right)|_{t_1=p'_1,\ldots,t_{k-1}=p'_{k-1}} \wedge \\ \left(care_j^{t_k} \wedge \mathit{diff}_j|_{t_k=1}\right)|_{t_1=p'_1,\ldots,t_{k-1}=p'_{k-1}}, \end{array}$$

for $i \neq j$. In this case, the two primary outputs f_i and f_j require t_k to be 1 and 0, respectively. It makes Eq. (9) satisfiable, and makes interpolation non-applicable. Note that the failure of interpolation is independent of the rectifiability of F. It may happen even when F is rectifiable.

To resolve the above problem, there can be two possible solutions:

1. **by pruning spurious counterexamples**: When the formula of Eq. (9) is found satisfiable under some satisfying assignment (x^*, t^*) to variable (X, T_k), the assignment t^* is blocked if it is not an actual counterexample to rectifiability. The process repeat until Eq. (9) is unsatisfiable.

2. **by taking the on-set or off-set function**: The on-set formula of Eq. (7) or the negation of the off-set formula of Eq. (8) can be directly taken as a patch function for p'_k.

In this work, we adopt the latter approach to generate an initial patch for both performance and patch quality considerations because interpolation computation may be expensive, especially when the underlying refutation proof is large. After an initial patch is generated, we will resort to interpolation-based synthesis for optimization as to be discussed.

5. LOCALIZATION FOR INITIAL PATCH SIMPLIFICATION

We propose a *localization* technique to improve the quality of initial patch generation. Empirical experience suggests that the proposed localization technique can dramatically reduce the runtime of interpolation-based patch optimization and substantially reduce patch sizes of difficult instances.

As mentioned in Section 4.3, we directly take ${p'}_k^{on}$ or $\neg {p'}_k^{off}$ for subsequent initial patch generation for target t_k. The so-derived initial patch depends on primary input variables X. Intuitively, we may improve the initial patch if it may depend on intermediate variables in addition to X. To achieve this, we extract subcircuits from the faulty circuit F and golden circuit G, which are relevant to rectification as shown in Algorithm 2. It takes three arguments, faulty circuit F, golden circuit G, and target signals T, as input. In line 2, FRAIG returns the set E of equivalence classes of signals among the two circuits. In line 3 (resp. line 4), procedure CUTFRONTIER returns a cut-frontier C_F (resp. C_G), which consists of a set of signals whose removal from F (resp. G) induces a separation on F (resp. G) into two subcircuits, one containing primary input signals and the other containing the primary output signals. The cut-frontier is found by a traversal in a reverse topological order from primary outputs to primary inputs by collecting the first found signal of type in {primary inputs X, signals in E, target signals in T} (resp. {primary inputs X, signals in E}) along every path from the primary outputs. In line 5, the union C_d of C_F and C_G is turned.

Algorithm 2 Localization
```
1: procedure LOCALIZATION($F, G, T$)
2:     $E \leftarrow$ FRAIG(F,G)
3:     $C_F \leftarrow$ CUTFRONTIER($F, X \cup E \cup T$)
4:     $C_G \leftarrow$ CUTFRONTIER($G, X \cup E$)
5:     $C_d \leftarrow C_F \cup C_G$
6:     return $C_d$
7: end procedure
```

After C_d is obtained by procedure LOCALIZATION, the following theorem allows us to generate an initial patch in terms of C_d.

THEOREM 2. *The on-set and off-set of the patch function* $p'_k(C_d, T_k)$ *can be expressed by*

$$\begin{aligned}{p'}_k^{on}(C_d, T_k) = \; & \bigvee_{j=1}^{m}(care_j^{t_k}(C_d, T \setminus \{t_k\}) \wedge \\ & diff_j(C_d, T)|_{t_k=0})|_{t_1=p'_1,\ldots,t_{k-1}=p'_{k-1}}\end{aligned} \quad (10)$$

$$\begin{aligned}{p'}_k^{off}(C_d, T_k) = \; & \bigvee_{j=1}^{m}(care_j^{t_k}(C_d, T \setminus \{t_k\}) \wedge \\ & diff_j(C_d, T)|_{t_k=1})|_{t_1=p'_1,\ldots,t_{k-1}=p'_{k-1}}\end{aligned} \quad (11)$$

The correctness of the theorem can be seen by the fact that all the output functions of F and G can be reexpressed in terms of the C_d signals.

6. COST OPTIMIZATION

We propose a heuristic algorithm to search minimal-cost patch solutions. We first show how to *rebase* (reselect a new set of base signals) a given patch from its initial base set, and then present our optimization method for cost reduction.

6.1 Rebasing with Functional Dependency

Given a patch function $p_k(B)$, where B is its initial bases. Our goal is to explore new bases for the patch function through functional dependency computation [12], and then synthesize a new patch function in terms of the new base.

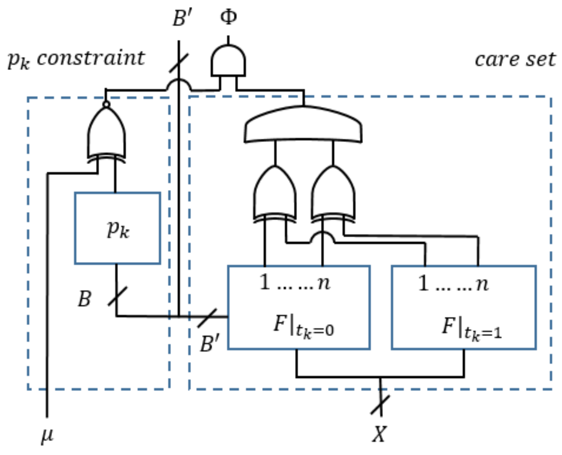

Figure 3: Half part of the rebase circuit.

The circuit shown in Figure 3 represents a part of our formula construction for functional dependency computation. In Figure 3, the left side, denoted by "p_k *constraint*," corresponds to the output constraint of patch function p_k, and the right side, denoted by "*care set*," is the circuit representing the care-set constraint.

Let the CNF (Conjunctive Normal Form) encoded formula of the circuit be $\Phi(\mu, B', X)$, where $B' = \{b_1, \ldots, b_\gamma\}$ is a set of candidate base signals, which can be intermediate or primary input signals of faulty circuit F. By adding some constraints and introducing control variables, our formula for rebasing with functional dependency exploration is as follows.

$$\begin{aligned}\Phi(\mu = 1, B', X) \wedge \Phi^*(\mu^* = 0, B'^*, X^*) \wedge \\ \textstyle\bigwedge_{1 \le i \le \gamma}(s_i \rightarrow b_i \equiv b_i^*).\end{aligned} \quad (12)$$

In Eq. (12), $\Phi(\mu, B', X)$ and $\Phi^*(\mu^*, B'^*, X^*)$ are two copies of the CNF formula of the circuit in Figure 3. These two formulas, $\Phi(\mu, B', X)$ and $\Phi^*(\mu^*, B'^*, X^*)$, correspond to the "A part" and "B part" of the *dependency logic network* constructed in [12], respectively. By introducing the selection variables s_i in Eq. 12, the subformula $(s_i \rightarrow b_i \equiv b_i^*)$ ensures that, if s_i is TRUE, then b_i and b_i^* are equivalent, that is, the signal b_i is selected in the base.

With Eq. (12), it requires the following two steps to check whether a candidate base set is feasible to implement the patch function p_k.

1. Determine a candidate base set.

2. Test whether the candidate base set is valid.

Step 1 can be done by exploiting the *unit assumption* mechanism provided by of a modern SAT solver, e.g., [6]. The enumeration of different base sets can be effectively achieved through incremental SAT solving by assigning different values to the selection variables s_i with the unit assumption interface. A candidate base set, determined by the truth assignment to the selection variables, is valid if and only if Eq. (12) is unsatisfiable under the corresponding unit assumptions on the selection variables. Note that if Eq. (12) is unsatisfiable under certain unit assumptions, the SAT solving may return an unsatisfiable subset of the unit assumptions that lead to the conflict. The information is useful to refine the base set.

With incremental SAT solving, we iteratively search for a better base set and maintain the best base found so far. Once a base set is determined, we may synthesize its corresponding patch function circuit by interpolantion [12].

6.2 Base Selection

In the previous section, Eq. (12) is used to test the feasibility of a candidate base and to resynthesize the patch function in terms of a new obtained base. Also with the help of Eq. (12), we propose our method for base selection to achieve cost reduction as follows.

Let B' be a set of possible base candidate signals. In our problem formulation, each $b'_i \in B'$ is associated with a constant weight, denoted $W(b'_i)$. Let $B \subseteq B'$ be the current base feasible for realizing the patch function $p_k(B)$, our method contains the following major steps:

1. Sort the base signals in B according to their weights in a non-increasing order. Put the first β sorted signals of B into a group called *Watch*; put the remaining signals in B into a group called *Hold*. The signal order is maintained in the two groups.

2. Keep signals in *Hold* being selected. We also select one at a time each candidate signal $b' \in \{B' \setminus \mathit{Hold}\}$ by setting its corresponding selection variable to TRUE in Eq. (12). Then run SAT solving to collect all counterexamples in terms of *Watch* variables. Let the set of counterexamples be $cex^{b'}$. (The counterexample enumeration process is to be detailed in Section 6.2.1.)

3. Repeatedly add the base candidate signal with the smallest value of the *cost per blocking* (CPB) into a set denoted as Γ, and run SAT solving, until the SAT solving result is unsatisfiable. Assuming the signals $b'_1, \ldots, b'_i$ are added to Γ in order, the CPB value of b'_i is define as
$$W(b'_i)/|cex_{i-1} \setminus cex^{b'_i}| \qquad (13)$$
where $cex_k = \bigcup_{j=1}^{k} cex^{b'_j}$. We then update *Hold* to be $\mathit{Hold} \cup \Gamma$, and empty Γ.

4. If the number of execution rounds equals the size $|B|$ of the initial base, then terminate. Otherwise, remove the first β signals from *Hold*, and replace signals in *Watch* by the β signals, and go to Step 3.

The intuition behind the above procedure is that the *Watch* signals are candidates to be removed from the current base and the *Hold* signals are to be kept. We intend to select some b' in the new base such that it can block as many counterexamples as possible that are not blocked by previously selected signals. Moreover, we want $W(b')$ to be small. The defined CPB score reflects the degree of non-preference of some b'.

Table 1: An example function for expressing counterexamples.

$a(X)$	$b(X)$	$p_k(a,b)$
0	0	0
0	1	1
1	0	1
1	1	0

Note that computing counterexamples for each candidate signal takes no more than $(2^{|\mathit{Watch}|} \times |B'|)$ SAT solving iterations. The runtime can be adjusted through the size of *Watch*. Our experience suggests that the setting $|\mathit{Watch}| = \beta = 5$ is a good trade-off between quality and performance.

6.2.1 Counterexample Enumeration

In Step 2 of the above procedure, to enumerate counterexamples with Eq. (12) in terms of the signals in *Watch*, we witness truth assignments to some variables in Eq. (12), and prune counterexamples with newly introduced control variables. Specifically, consider a patch function $p_k(a, b)$ shown in Table 1, which is driven by two signals a and b. In this case, $\Phi(\mu = 1, B', X)$ and $\Phi^*(\mu^* = 0, B'^*, X^*)$ in Eq. (12) characterize the on-set and off-set of $p_k(a, b)$, respectively. For a and b being the base signals of Φ, we let a^* and b^* be the base signals of Φ^*. If we do not select any base (including a and b), then Eq. (12) will be `satisfiable`. Its counterexamples can have following configurations:
$$\mathit{on} \times \mathit{off} = \{(ab, a^*b^*)|(01, 00), (01, 11), (10, 00), (10, 11)\},$$
where *on* and *off* are on-set and off-set of $p_k(a, b)$, respectively. Assume that we now witness a counterexample, say $(01, 00)$; we add clause
$$(c_1 \to a \vee \neg b),$$
where c_1 is a newly introduced control variable, to the SAT solver. Then we set c_1 to TRUE to escape from $(01, a^*b^*)$ in the following SAT solving. The next generated counterexample can be $(10, 00)$ or $(10, 11)$, we add again clause
$$(c_2 \to \neg a \vee b)$$
to the SAT solver. Now with c_2 being set to TRUE, the subsequent SAT solving will return `unsatisfiable`. It indicates that we have enumerated all counterexamples in terms of a and b.

Note that in Step 2 of the above procedure, each base candidate signal $b' \in B'$ is selected one at time for its counterexample collection. The control variables added for counterexample collection of b' should be set to FALSE when the counterexamples of a new base candidate signal $b'' \in B'$ are to be collected.

7. EXPERIMENTAL RESULTS

Our algorithm was implemented in the *ABC* [2] environment. The benchmark suite is from the 2017 ICCAD CAD Contest [11]. The experiments were conducted in single thread on a virtual machine with a 2.3 GHz Intel CPU, 4MB cache, and 64GB memory.

Our experience suggests that, directly apply previous methods [4][19] of multi-ECO could timeout in difficult instances such as units 6, 10, 11 and 19. Therefore, we compare our algorithm with state-of-the-art method in the 2017 ICCAD CAD Contest. Table 2 compares our results against those

Table 2: Comparison on cost, patch size, and runtime.

		1st place winner in [9]			ours			ratio		
ckt name	#target	cost	size	time	cost	size	time	cost	size	time
unit 1	1	4	1	0.17	4	1	0.17	1.00	1.000	1.00
unit 2	1	17	4	0.62	17	5	2.08	1.00	1.250	3.35
unit 3	1	80	3	0.17	80	3	0.32	1.00	1.000	1.88
unit 4	1	42	5	0.17	36	2	0.22	0.86	0.400	1.29
unit 5	2	47	30	16.43	47	49	13.5	1.00	1.633	0.82
unit 6	2	5660	6605	112.3	**118**	**5**	**36.29**	0.02	0.001	0.32
unit 7	1	284	2	4.33	284	2	9.73	1.00	1.000	2.25
unit 8	1	78	4	3.89	80	5	39.29	1.03	1.250	10.10
unit 9	4	50	29	1.17	50	50	8.18	1.00	1.724	6.99
unit 10	2	135	587	52.68	135	**310**	**9.84**	1.00	0.528	0.19
unit 11	8	4142	1063	543.29	**760**	**369**	**229**	0.18	0.347	0.42
unit 12	1	104	1	0.52	104	1	1.12	1.00	1.000	2.15
unit 13	1	3467	9	0.57	3833	12	6.83	1.11	1.333	11.98
unit 14	12	95	42	12.64	104	65	74.07	1.09	1.548	5.86
unit 15	1	191	11	1.53	168	5	3.53	0.88	0.455	2.31
unit 16	2	318	13	17.57	260	13	5.63	0.82	1.000	0.32
unit 17	8	434	79	2.33	436	101	20.6	1.00	1.278	8.84
unit 18	1	18	1	6.49	106	21	58.2	5.89	21.000	8.97
unit 19	4	501804	7686	217.4	**15532**	**13**	**42.98**	0.03	0.002	0.20
unit 20	4	136	6	0.77	120	6	16.25	0.88	1.000	21.10
geo. mean								**0.68**	**0.56**	**2.00**

of the first place winner in the 2017 ICCAD CAD Contest. (Official announcement and results can be found in [9], and [10].) The first column shows the testcases. Column 2 lists the number of target signals to be rectified; Columns 3 and 6 list the patch cost; Columns 4 and 7 list the gate counts of the patch with all primitive gates are of size one; Columns 5 and 8 list the CPU time in seconds; the last three columns list the ratios of the results of the contest winner to ours in terms of cost, size, and runtime. The last row reports the geometric means of the three compared metrics.

As can be seen, our method outperforms the first place winner in the 2017 ICCAD CAD Contest in highlighted difficult ECO instances, in terms of patch cost, size and runtime. Specifically, the cost we obtained has 47x, 5x, and 32x reduction in unit 6, unit 11, and unit19 respectively. Regarding the patch size, our result has 1321x, 1.8x, 2.8x, and 591x reduction in unit 6, unit 10, unit11 and unit 19 respectively. Our runtimes are also shorter in these difficult ECO instances.

Besides difficult ECO instances, our runtimes are longer in several cases. For example, in unit 14, it takes longer time to finish the optimization of patch. But the overall results are comparable to the first place winner in the 2017 ICCAD CAD Contest. The geometric means of the patch cost and size ratios are 0.68 and 0.56, respectively.

8. CONCLUSIONS

In this paper, we presented an algorithm for solving the multi-fix ECO problem. In addition to cost optimization techniques, we proposed a localization method that significantly improves the patch quality especially for difficult ECO instances. The overall performance of our method outperforms the top winning team in the 2017 CAD Contest in terms of both patch cost and patch size while the runtime maintains reasonable.

Acknowledgments

This work was supported in part by the Ministry of Science and Technology of Taiwan under grants 104-2628-E-002-013-MY3, 105-2221-E-002-196-MY3, and 105-2923-E-002-016-MY3.

9. REFERENCES

[1] V. Balabanov and J.-H. R. Jiang. Unified QBF Certification and Its Applications. *Formal Methods in System Design*, 41(1): 45-65, 2012.

[2] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. Available: `http://www.eecs.berkeley.edu/∼alanmi/abc/`

[3] H.-Y. Chang, H.-R. Jiang, and Y.-W Chang. Simultaneous Functional and Timing ECO. In *Proc. DAC*, pp. 140-145, 2011.

[4] A.-C. Cheng, H.-R. Jiang, and J.-Y. Jou. Resource-aware functional ECO patch generation. In *Proc. DATE*, pp. 1036-1041, 2016.

[5] W. Craig. Linear Reasoning: A New Form of the Herbrand-Gentzen Theorem. *J. Symbolic Logic*, 22(3): 250-268, 1957.

[6] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. SAT*, pp. 502-518, 2003

[7] K.-H. Ho, J.-H. R. Jiang, and Y.-W. Chang. TRECO: Dynamic Technology Remapping for Timing Engineering Change Orders. *IEEE Trans. on CAD*, 31(11): 1723-1733, 2012.

[8] S.-L. Huang, W.-H. Lin, and C.-Y. Huang. Match and Replace: A Functional ECO Engine for Multi-error Circuit Rectification. In *Proc. ICCAD*, pp. 383-388, 2011.

[9] C.-J. Hsu, C.-A. Wu and C.-Y. Huang, "Resource-aware patch generation," ICCAD CAD Contest, Problem A, 2017. `http://cad-contest-2017.el.cycu.edu.tw/CAD-contest-at-ICCAD2017/`

[10] C.-J. Hsu, C.-A. Wu and C.-Y. Huang, Presentation of "Resource-aware patch generation," ICCAD CAD Contest, Problem A, 2017. `http://cad-contest-2017.el.cycu.edu.tw/Problem%20A_Ching-Yi.pdf`

[11] C.-Y. Huang, C.-J. Hsu, C.-A. Wu, and K.-Y. Khoo. ICCAD-2017 CAD Contest in resource-aware patch generation. In *Proc. ICCAD*, pp. 857-862, 2017.

[12] 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.*, pp. 457-467, 2010.

[13] J.-H. R. Jiang, H.-P. Lin, and W.-L. Hung. Interpolating functions from large Boolean relations. In *Proc. ICCAD*, pp. 779-784, 2009.

[14] S. Krishnaswamy, H. Ren, N. Modi, and R. Puri. DeltaSyn: An Efficient Logic Difference Optimizer for ECO Synthesis. In *Proc. ICCAD*, pp. 789-796, 2009.

[15] C.-C. Lin, K.-C. Chen, S.-C. Chang, M. Marek-Sadowska, and K.-T. Cheng. Logic Synthesis for Engineering Change. In *Proc. DAC*, pp. 647-652, 1995.

[16] K. L. McMillan. Interpolation and SAT-based model checking. In *Proc. CAV*, pp. 1-13, 2003.

[17] A. Mishchenko, R. K. Brayton, J.-H. R. Jiang, and S. Jang. Scalable don't-care-based logic optimization and resynthesis *ACM Trans. Reconfigurable Technology and Systems*, 4(4), article no. 34, 2011.

[18] A. Mishchenko, S. Chatterjee, J.-H. R. Jiang, and R. K. Brayton. FRAIGs: A unifying representation for logic synthesis and verification. *ERL Technical Report*, 2005.

[19] 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. DAC*, pp. 146-151, 2011.

[20] K.-F. Tang, P.-K. Huang, C.-N. Chou, and C.-Y. Huang. Multi-Patch Generation for Multi-Error Logic Rectification by Interpolation with Cofactor Reduction. In *Proc. DATE*, pp. 1567-1572, 2012.

[21] 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. ICCAD*, pp. 729-734, 2010.