Efficient Computation of ECO Patch Functions

Ai Quoc Dao[1], Nian-Ze Lee[2], Li-Cheng Chen[2], Mark Po-Hung Lin[1], Jie-Hong R. Jiang[2], Alan Mishchenko[3], and Robert Brayton[3]

[1] Department of EE & AIM-HI, National Chung Cheng University, Chiayi, Taiwan
[2] Department of EE, National Taiwan University, Taipei, Taiwan
[3] Department of EECS, University of California, Berkeley, CA, USA

aiquocvt@gmail.com, {d04943019, r05943146}@ntu.edu.tw, marklin@ccu.edu.tw, jhjiang@ntu.edu.tw, {alanmi, brayton}@berkeley.edu

ABSTRACT

Engineering Change Orders (ECO) modify a synthesized netlist after its specification has changed. ECO is divided into two major tasks: finding target signals whose functions should be updated and synthesizing the patch that produces the desired change. This paper proposes an efficient SAT-based solution for the second task: resource-aware computation of multi-output patch functions. The solution is based on several new algorithms and outperforms the top three winners of the 2017 ICCAD CAD Contest (Problem A).

1. INTRODUCTION

In a typical hardware design flow, synthesis of the design begins while changes are still being made to the design specification. In this case, a synthesized netlist may need to be updated to reflect recent changes. In another scenario, a bug is detected late in the design flow and an already synthesized design has to be incrementally modified to accommodate the bug fix. In both cases, it is faster and less costly to update the synthesized netlist rather than to perform synthesis from scratch.

Engineering Change Order (ECO) [4,5,7,9,11,13-15] is the task of incrementally updating an implementation of the design when its specification has changed. ECO research started in the 1990s and has matured to the point where major EDA vendors offer automated solutions [17,18].

The task of performing ECO can be divided into several subtasks: (1) isolating one or more design blocks whose functionality is to be changed, (2) detecting target signals where the changes should be applied, (3) computing new Boolean functions (the patch functions) of the target signals, (4) verifying that the netlist is equivalent to the specification when patch functions are inserted at the target signals in the implementation.
This paper focuses on the most computationally challenging aspect of ECO, which is efficiently solving task (3) for multiple target signals. The proposed solution is SAT-based. The computation is performed as a sequence of carefully orchestrated SAT calls. It scales to large netlists and results in small patch functions using intermediate signals. In the case where SAT-based computations time out, a low-effort structural analysis is used to derive a patch in terms of primary inputs.

DAC '18, June 24–29, 2018, San Francisco, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196039

The proposed algorithm involves several components:

- **structural pruning** to narrow down the scope of the netlist where the changes have to be applied,
- **patch support computation** to detect a minimal-cost subset of signals sufficient to implement the patch,
- **patch function computation** to derive Boolean function(s) that, when used in the implementation netlist, make it equivalent to the specification.

The contributions of this paper are in the following improvements to the ECO solution from previous work [15]:

- extending ECO to multiple target signals,
- extending support computation to be cost-aware,
- support computation based on an exact algorithm for single target fix and a heuristic algorithm with complexity reduced from N up to $log(N)$ SAT calls, where N is the number of candidate signals,
- faster computation of patch functions using cube-enumeration rather than general interpolation,
- structural computation of patch functions in terms of primary inputs useful when the SAT solver times out
- method for patch quality improvement based on a quantified Boolean formula (QBF) and maximum-flow.

The proposed combinational ECO solution can be extended to be sequential as shown in [10].

The rest of the paper is organized as follows. Section 2 contains necessary background. Section 3 gives an overview of the algorithm. Section 4 discusses experimental results. Section 5 concludes the paper.

2. BACKGROUND

2.1 Boolean Function

In this paper, *function* refers to a completely specified Boolean function $f(X)$: $B^n \rightarrow B$, $B = \{0,1\}$. The *support* of function f is the set of variables X, which influence the output value of f. The support size is denoted by $|X|$.

2.2 Boolean Network

A *Boolean network* (or *circuit*) is a directed acyclic graph (DAG) with nodes corresponding to Boolean functions and edges corresponding to wires connecting the nodes.

A node n may have zero or more *fanins*, i.e. nodes driving n, and zero or more *fanouts*, i.e. nodes driven by n. The primary inputs (PIs) are nodes without fanins. The primary outputs (POs) are a subset of nodes of the network, connecting it to the environment. A *transitive fanin (fanout) cone* (TFI/TFO) of a node is a subset of nodes of the network, which are reachable through the fanin (fanout) edges of the node. The *TFO support* of a node is the set of primary outputs in the TFO of the node.

2.3 Boolean Satisfiability

A *satisfiability problem* (SAT) is a decision problem that takes a propositional formula representing a Boolean function and answers the question whether the formula is satisfiable, that is, whether the Boolean function is not a constant 0. The formula is *satisfiable* (SAT) if there is an assignment of variables such that the formula takes value 1. Otherwise, the propositional formula is *unsatisfiable* (UNSAT). A software program that solves SAT problems is called a *SAT solver*. When a SAT problem is satisfiable, the solver returns a satisfying assignment, which is often useful to reproduce and understand the malfunction.

2.4 Conjunctive Normal Form (CNF)

To derive a propositional formula for the SAT solver, important aspects of the problem are encoded using Boolean *variables*. Presence or absence of an aspect is represented by a positive or negative *literal* of the variable. A disjunction of literals is called a *clause*. A conjunction of clauses is called a *CNF*. The CNF can be processed by a mainstream CNF-based SAT solver, such as MiniSAT [6] used in this work.

2.5 SAT-Based ECO Formulation

The ECO problem considered in this paper is formulated by providing two netlists called *implementation* and *specification*. The netlists have the same number of inputs and outputs but different functionality. Additionally, the implementation netlist contains one or more nodes called *targets* (aka rectification signals [15]). The ECO solution achieves the following goals:

- It results in a set of *patch functions* which, when used as new local functions of the target nodes, make the implementation netlist functionally equivalent to the specification netlist.
- The support of the patch functions is selected among the nodes in the implementation netlist. Each support node has a cost associated with it. The higher the cost, the less desirable is the node as an input of the patch. The ECO solution minimizes the cost of the support of the resulting patch.
- The ECO solution minimizes the number of gates used in the patch functions.
- Finally, the ECO solution is robust in that it solves many practical problems in a matter of seconds.

2.5.1 Previous Work

In this subsection, we summarize the SAT-based solution for the single-target ECO problem proposed in [15].

In the following sections, we will improve and extend this solution to handle multiple targets, compute minimal-cost supports, and compute patch functions by cube enumeration rather than interpolation. These changes make ECO more efficient and constitute the main contribution of this paper.

Figure 1 shows the ECO miter constructed to compare the Boolean functions of the implementation network and the specification network. The implementation side contains the target node, n. The resulting miter, $M(n, x)$, produces value 1 when the implementation and the specification differ for at least one output pair for the input x.

The ECO problem is elegantly summarized using the following quantified Boolean formula:

$$\forall x \exists n\ [F(n, x) = S(x)].$$

This expression reads as follows: ECO has a solution if, for any input minterm x, there exists a value of the target node n, such that the implementation is equivalent to the specification.

By complementing this formula and using $M(n, x)$ to denote the Boolean function of the ECO miter, we can say that, for the ECO solution to exist, the following expression should be false:

$$\exists x \forall n\ [F(n, x) \neq S(x)] = \exists x \forall n\ M(n, x). \qquad (1)$$

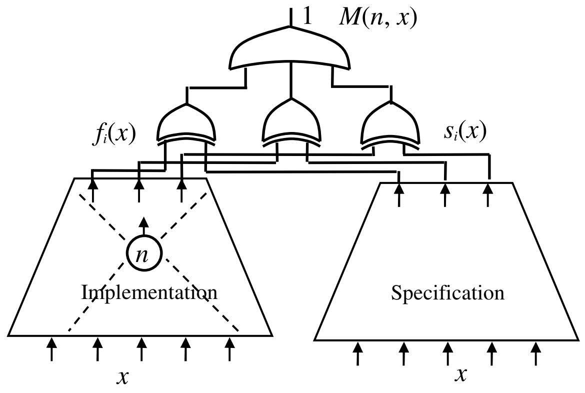

Figure 1: The ECO miter.

2.5.2 Patch in Terms of Primary Inputs

Consider the conjunction of the two cofactors of the miter, $M(0, x)$ and $M(1, x)$, w.r.t. the target signal n, which corresponds to universally quantifying n in (1).

If expression $M(0, x)\ \&\ M(1, x)$ is not a constant 0, it means that there exists a minterm x, such that the difference between implementation and specification is observed for both values of the target n. In this case, ECO is impossible because, no matter what value the target takes for this minterm x, the functional difference remains.

If, on the other hand, expression $M(0, x)\ \&\ M(1, x)$ is equivalent to a constant 0, we can compute interpolant $I(x)$, which is a Boolean function, such that $M(0, x) \Rightarrow I(x)$ and $I(x) \Rightarrow\ !M(1, x)$, and use this function as the patch function. This makes the implementation equivalent to the specification. Indeed, $I(x)$ produces value 1 for all minterms x in $M(0, x)$, thus fixing all the mismatches that occurred under $n = 0$. Similarly, $I(x)$ produces value 0 for all minterms x in $M(1, x)$, thus fixing all the mismatches that occurred under $n = 1$. This proves that the resulting patch function solves the ECO problem.

2.5.3 Patch in Terms of Internal Nodes

The above discussion assumes that the patch depends on primary inputs. However, in practice, it is helpful to express the patch in terms of intermediate variables, possibly including some primary inputs. This often leads to smaller patches as well as patches depending on variables that are close to the target if an initial placement is given.

In this case, we conjoin the miter $M(n, x)$ with the candidate divisor relation $R(d, x) = [d == D(x)]$, which relates the selected set of divisor variables d with their functions $D(x)$ in terms of primary inputs, resulting in an extended miter, $M(n, x)\ \&\ R(d, x)$. We consider two copies of the extended miter in terms of variables x_1 and x_2. The following expression

$$M(0, x_1)\ \&\ M(1, x_2)\ \&\ R(d, x_1)\ \&\ R(d, x_2) \qquad (2)$$

corresponds to Figure 5 in [15]. If the expression is satisfiable, ECO is not possible. A likely reason for this is that the set of divisors used in the relation $R(d, x)$ is not sufficient to express the patch. On the other hand, if the expression is unsatisfiable, ECO exists. The patch function can be computed as an interpolant in terms of the d variables, which are the common variables in the following unsatisfiable SAT instance:

$$[M(0, x_1) \ \& \ R(d, x_1)] \ \& \ [M(1, x_2) \ \& \ R(d, x_2)]. \qquad (3)$$

We remark that while implementing the expression (3), since the functions $D(x)$ of divisor variables d exist in the miter $M(n,x)$, no explicit conjunction between $M(n,x)$ and $R(d,x)$ is performed. Instead, an auxiliary variable a is assigned to each pair of candidate divisor variables d_1 and d_2 in $M(0, x_1)$ and $M(1, x_2)$, respectively, and a constraint (a' OR ($d_1 = d_2$)) is added into the SAT solver. When the auxiliary variable a is assumed to be 1, the constraint forces d_1 equal to d_2, and hence they become common variables in the above UNSAT instance.

3. PROPOSED ALGORITHM

This section describes the proposed algorithm in detail. A high-level overview is shown in Figure 2.

The algorithm takes two netlists, implementation and specification, with the same inputs and outputs. The implementation netlist has a set of nodes labeled as targets. There are no additional requirements for the specification netlist, in particular, it is not expected that it has any structural similarity with the implementation netlist. Detailed descriptions of verifying the resulting patch and all computation of ECO patch functions are given in the subsections below.

3.1 Handling Multiple Targets

Expression (1) is a general ECO formulation, which works for any number of targets, but the solution discussed above assumes one target. Rather than developing a dedicated multi-target ECO solution, we reduce the multi-target case to an iterative ECO computation for one target.

The proposed computation considers targets one at a time. For target n_i, all other targets, except this one, are universally quantified from the circuit representation of the miter $M(n, x)$, resulting in the updated miter $M_i(n_i, x)$. The one-target ECO problem with the updated miter is solved as discussed above, resulting in the patch for target n_i. Next, this patch is substituted into the original miter and the other targets are processed similarly until all have been treated.

By considering the ECO problem one target at a time, we incrementally build each patch with the following property: the patch solves the ECO for those primary input minterms, for which the problem could not be solved using the quantified targets. In a way, these are essential minterms of the ECO problem, which can only be fixed using the current target.

Theorem 1: Expression (1) is UNSAT if and only if the above sequence of one-target ECO problems has a solution at each step.

Proof:

➔ Note that a QBF is UNSAT iff it has a Herbrand function countermodel [1], and that the patch function derived for target n_i in the above sequence is a Herbrand function of variable n_i for the above QBF. Essentially the procedure is equivalent to performing ∀-quantifier elimination on one variable n_i at a time, by sequentially substituting it with its Herbrand function to eliminate variable n_i from the QBF. By the existence of a Herbrand function countermodel, the above sequence of one-target ECO problems must have a solution at each step.

🡐 Assume by contradiction that the above sequence yields a solution at each step and the QBF is SAT. By [1], no Herbrand function countermodel exists for a SAT QBF. It contradicts the assumption that the sequence yields a solution, i.e., Herbrand function, at each step. ■

A more detailed formulation of Theorem 1 is found in [20].

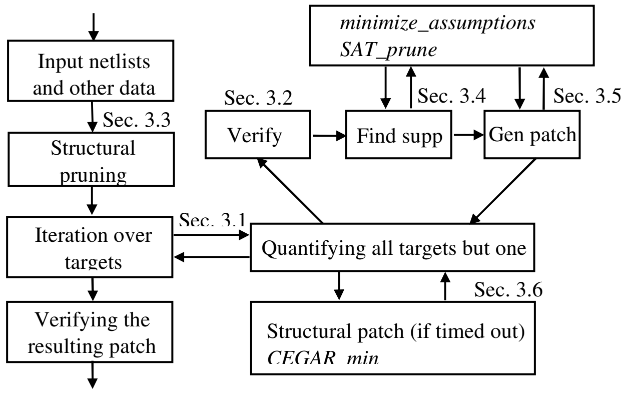

Figure 2: Overview of the algorithm.

3.2 Verifying That The Set of Targets Is Sufficient

The computation begins by verifying that the set of targets is sufficient to solve the ECO problem.

For this, we evaluate expression (1) in Section 2.5. We universally quantify the target signals and use the SAT solver to check satisfiability of the resulting circuit using combinational equivalence checking [12].

If the SAT problem is satisfiable, the ECO has no solution. If the SAT problem is unsatisfiable, the ECO has a solution, that is, for each target, there exist a patch in terms of primary inputs and internal variables (by the construction of [15]); when the targets are replaced by their patches, the updated implementation is equivalent to the specification.

If the SAT solver times out, the status of the ECO problem is not known. In this case, the solution is assumed to exist and the structural method in Section 3.6 is used to derive the patch in terms of primary inputs. If the timed-out SAT problem is in fact UNSAT, the computed patch is correct. Otherwise, the ECO has no solution, and the patch is invalid.

As an alternative, we also evaluate the QBF instance described by expression (1) directly using command *qbf* in ABC. If the instance is UNSAT, the ECO problem has a solution, and the information gathered during QBF solving is used to improve the quality of the structural patch, as detailed in Section 3.5.

3.3 Structural Pruning

Structural pruning computes a logic window used while solving the ECO problem. The following computations are performed:

- Traverse TFO of the targets in the implementation netlist and mark POs reachable from the targets. These signals are POs of the window.
- Traverse TFI of these POs in both implementation and specification and compute the PIs reachable in both netlists. These signals are PIs of the window.
- Find all signals in the implementation, which are not in the TFO of the targets and whose support is contained in

the computed set of PIs. These signals are candidate divisors for expressing patch functions.

The miter in Figure 1 includes all logic nodes between the window PIs and window POs, computed as described above. By construction, these nodes include targets and candidate divisors.

3.4 Patch Support Computation

We present two algorithms to compute support signals for a patch, respectively, in the following sections.

3.4.1 Fast Minimization of Divisors

As shown in Section 2.5.3, if the set of candidate support signals (called *divisors)*, *d*, is sufficient to generate a valid patch, expression (2) is unsatisfiable. The patch support computation finds a feasible low-cost support of the patch using a novel SAT-based procedure *minimize_assumptions* shown in Algorithm 1. This procedure is closely related to the novel SAT-based method LEXUNSAT [19], which computes a minimal canonical subset of divisors sufficient to express a given Boolean function.

The procedure takes a CNF representation of function *F* and the array *A* of assumptions. It returns the size, *S*, of the minimized subset of *A*. The assumptions in *A* are reordered so that the first *S* assumptions in *A* after calling this procedure represent the minimized subset.

The patch support computation works by instrumenting the CNF representing expression (2) with an auxiliary variable for each candidate divisor. When an auxiliary variable is assumed to be 0, the divisor is not used in the patch. When an auxiliary variable is 1, the divisor is used.

Positive literals of the auxiliary variables are collected in the array in increasing order of the costs of corresponding divisors. These literals are assumptions given to procedure *minimize_assumptions.* The procedure returns a minimized subset with the following properties: (1) the subset is minimal, that is, no divisor can be removed without compromising the existence of ECO; (2) the sum total of costs of divisors included in the subset is minimized, i.e. a divisor is included in the subset, if it cannot be replaced by one with a lower cost.

This procedure is fast and results in good quality of divisors. This is because of the minimality properties mentioned above, and because the complexity is $O(\max \{\log(N), M\})$ where *N* is the number of divisors considered, and *M* is the number of selected divisors in the final support set. This is better than the complexity $O(N)$ of the naïve implementation, which tries adding divisors one at a time in the given order.

It should be noted that the resulting patch is minimal but not minimum. It is sometimes possible to improve the patch cost by making greedy changes. To this end, a last-gasp patch-improvement step is performed when we try replacing one support variable with another if it has a lower cost.

3.4.2 SAT-Based Exact Pruning

As the above subroutine *minimize_assumptions* only guarantees to produce a patch of minimal, rather than minimum, cost, we further propose a *SAT-based exact pruning* method, abbreviated as *SAT-prune*, to obtain a minimum-cost patch for one target rectification.

The procedure initiates a new SAT solver in order to iteratively prune the search space by adding new clauses in two ways: one is to block divisors whose cost cannot be smaller than the current minimum, and the other is to block infeasible divisors. The pruning terminates when the solver returns UNSAT, which denotes that all combinations of divisors have been examined and the solution with minimum cost has been found.

3.5 Patch Function Computation

Computation of the patch minimizes the circuit implementing the patch function with a known support.

The Sum-Of-Products (SOP) representing the patch function, in terms of the divisors *d,* is computed by enumerating satisfying assignments of the extended miter, $M(n, x)$ & $R(d, x)$, and expanding them into prime cubes.

```
int minimize_assumptions( function F, assumptions A )
{
    // if there is only one assumption in A, check if it is needed
    if ( |A| == 1 ) {
        if ( F is UNSAT without assuming A )
            return 0; // this assumption is not needed
        else
            return 1; // this assumption is needed  }
    // divide assumptions into two parts: lower and higher
    A_low = entries in A with index in range [0; |A|/2];
    A_high = entries in A with index in range [|A|/2+1; |A|-1];

    // try the lower part A_low without the higher part A_high
    if ( F is UNSAT while assuming A_low )
        return minimize_assumptions( F, A_low );
    // find solution for A_high while assuming A_low
    incrementally assume all assumptions in A_low;
    S_high = minimize_assumptions( F, A_high );
    incrementally unassume all assumptions in A_low;
    // reorder assumptions
    place S_high entries in A_high before all entries in A_low;
    // find solution for A_low assuming part of A_high is used
    incrementally assume S_high first assumptions in A_high;
    S_low = minimize_assumptions( F, A_low );
    incrementally unassume S_high assumptions in A_high;

    // the result is composed of assumptions used in both parts
    return S_high + S_low;
}
```

Algorithm 1: Pseudo-code of procedure *minimize_assumptions*.

This is achieved by iteratively employing the SAT solver containing the CNF of the extended miter:

- assume $n = 0$ and get a satisfying assignment; if there is no such assignment, quit the iteration;
- assume $n = 1$ and use assumptions composed of the complements of each *d*-literal in the satisfying assignment;
- input this array to the *minimize_assumptions* procedure shown in Section 3.3 (the procedure returns a minimal subset of assumptions, which is a prime cube in *d*-variables);
- block this cube in the on-set of the extended miter and continue with the next assignment.

When the above iteration terminates, an irredundant prime SOP representation of the patch function is obtained. The SOP expression is then factored and synthesized in ABC [16]. The resulting multi-level circuit is returned as the patch function.

3.6 Structural Patch Computation

If the SAT-based feasibility computation times out, a patch in terms of primary method is derived structurally.

3.6.1 Single Target Fix

To apply this method, observe that the negative cofactor of the ECO miter, $M(0, x)$, is a single-output Boolean function in terms of primary inputs. It be used as a patch function for the target node because $M(0, x)$ is an interpolant of $M(0, x)$ & $M(1, x)$, if this expression is unsatisfiable. This allows us to derive the patch structurally without the SAT solver. We use a circuit for $M(n, x)$ and assume $n = 0$. The resulting circuit, after synthesis, can be used as a patch function in terms of primary inputs.

3.6.2 Multiple Target Fix

The construction of structural patches in the case of multiple targets can be done by cofactoring the ECO miter at the target signals with different values through the certificate generation method for CEGAR-based QBF solving [2]. The technique is particularly useful when the number of targets is large. For example, the number of copies of ECO miters needed to construct a structural patch for an implementation with 8 targets is reduced from 255 to 40 by leveraging information from QBF solving.

3.6.3 Replacing Primary Inputs by Other Nodes

The size of the patch might be reduced further by performing Boolean resubstitution of the internal divisor variables into the circuit representation of the patch. The resubstitution can be done structurally or functionally. Although the latter requires SAT solving, the SAT queries are simpler than when computing the patch support. This is because only the implementation, rather than the whole ECO miter, is used for checking resubstitution.

On the other hand, the resubstitution can be done structurally, using *maximum flow* computation. Given an implementation $F(n, x)$ and a patch circuit $n(x)$ in terms of primary inputs x, we find equivalent signals between $F(n, x)$ and $n(x)$. In $F(n, x)$, signals having equivalent counterparts in $n(x)$ will form a cut. We can use the maximum-flow/min-cut algorithm to find a min-cut consisting of signals whose summation of weights is minimized. The signals on the min-cut are used as new supports of the patch. This method, abbreviated as *CEGAR_min* in the subsequent discussion, helps improve quality of results (QoR) of the structural patches when SAT-based methods are unavailable.

4. EXPERIMENTAL RESULTS

The proposed algorithms are implemented in ABC environment, and the experiments are conducted on a Linux machine with Intel Xeon 2.1 GHz CPU and 126 GB RAM.

4.1 2017 ICCAD CAD Contest Benchmarks

The benchmarks used in our experiments are provided by the contributors of the contest Problem A and available online [8]. These benchmarks were created using ISCAS-85/89 [3], ITC-99, IWLS-2005, OpenCore, and LGSynth-93 benchmarks as well as real-world ECO problems coming from industrial designs.

The benchmark suite includes 20 resource-aware ECO instances, each containing a netlist of the old implementation, a netlist of the new specification, and a file with the weight of each signal in the old implementation.

To reflect multiple considerations during the process of ECO, the contest benchmark used 8 different weight distributions:

- Type T1 (Distance-aware distribution-A): The nodes' weights become larger when they are closer to PIs in some parts of the circuits.
- Type T2 (Distance-aware distribution-B): The nodes' weights become larger when they are farther away from PIs in some parts of the circuits.
- Type T3 (Path-aware distribution): The nodes on some paths have larger weights than the others in parts of the circuits.
- Type T4 (Locality-aware distribution): The nodes on some parts of the circuit have larger weights than the others.
- Type T5: Composed of T1 and T3.
- Type T6: Composed of T2 and T3.
- Type T7: Composed of T1 and T4.
- Type T8: Highly mixed and undulating distribution.

4.2 Results of Proposed Algorithms

The statistics of the contest benchmark suit and the results of the proposed algorithms are shown in Table 1. Columns 2, 3, 4, 5, and 6 show the numbers of PIs, POs, gates in the old implementation, gates in the new specification, and target point(s) in the old implementation for each benchmark. Columns 7, 8, and 9 show the resource cost, the size of the patch, and the runtime in seconds, respectively, of the proposed algorithm without applying the subroutine *minimize_assumptions*. Instead *analyze_final* [6] was used after applying the SAT solver to the unsatisfiable expression (2). These results are used to evaluate the role of *minimize_assumptions*, *SAT_prune*, and *CEGAR_min*. Columns 10, 11, and 12 (resp. 13, 14, and 15) show the resource cost, the size of the patch, and the runtime in seconds of the proposed algorithm with *minimize_assumptions, which is the 1st* place in 2017 CAD Contest (resp. *SAT_prune* and *CEGAR_min*).

As shown in Table 1, the subroutine *minimize_assumptions* achieves an average of 74% and 53% reduction, respectively, on the cost and the size of a patch, with an acceptable overhead in runtime, compared to the baseline. On the other hand, with *SAT_prune* and *CEGAR_min*, the ratios of cost and patch size reductions are further improved to 76% and 57%, respectively.

As mentioned in Section 3.4.2, *SAT_prune* guarantees a cost-minimum patch when the implementation has one target. Therefore, for single-target benchmark such as unit13, compared to the results of *minimize_assumptions*, the *SAT_prune* method further reduces the resource cost from 3467 to 2656.

On the other hand, for benchmarks with multiple targets, *SAT_prune* might be trapped in a local optimum while fixing the first target, and miss the global optimum. Therefore, in benchmarks unit9, and unit17, the costs found by *SAT_prune* are worse than those found by *minimize_assumptions*.

Benchmarks unit6, unit10, unit11, and unit19 are solved by the structural method. As shown in Table 1, both the cost and the size of a structural patch are improved by *CEGAR_min*.

From our observations from the experimental results over CAD contest benchmarks suit, the subroutine *minimize_assumptions* provides a scalable solution with good QoR; to trade scalability for QoR, *SAT_prune* guarantees the cost-minimum result under the single-target scenario. When both methods fail, *CEGAR_min* improves QoR of structural patches. Our results outperform the winners of the 2017 ICCAD CAD Contest [8].

5. CONCLUSIONS

This paper presents an efficient SAT-based method for resource-aware computation of patch functions in combinational ECO with multiple targets. The proposed solution is based on several new algorithms and heuristics. Experimental results indicate that it works well in practice.

Future work will include creating an integrated ECO flow in ABC, which detects a set of target nodes, followed by applying the proposed patch computation when the targets are known.

ACKNOWLEDGEMENTS

This work was supported in part by SRC contract 2710.001 “SAT-based methods for scalable synthesis and verification”, NSF/NSA grant “Enhanced equivalence checking in cryptanalytic applications”, and the Ministry of Science and Technology (MOST), Taiwan, under Grant No. 104-2628-E-194-001-MY3 and 105-2221-E-002-196-MY3.

REFERENCES

[1] V. Balabanov and J.-H. Jiang, “Unified QBF certification and its applications,” *FMSD*, 2012.

[2] V. Balabanov, J.-H. Jiang, C. Scholl, A. Mishchenko, and R. Brayton, “2QBF: Challenges and solutions,” in *Proc. SAT*, 2016.

[3] F. Brglez, D. Bryan, and K. Kozminski, “Combinational profiles of sequential benchmark circuits,” in *Proc. ISCAS*, 1989.

[4] K.-H. Chang, I. Markov, and V. Bertacco, “Fixing design errors with counterexamples and resynthesis,” in *Proc. ASPDAC,* 2007.

[5] A.-C. Cheng, H.-R. Jiang, and J.-Y. Jou, “Resource-aware functional ECO patch generation,” in *Proc. DATE*, 2016.

[6] N. Een and N. Sörensson, “An extensible SAT-solver,” in *Proc. SAT*, 2003.

[7] 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 TCAD*, 2013.

[8] C.-Y. Huang, C.-J. Hsu, and C.-A. Wu, “2017 ICCAD CAD Contest Problem A: Resource-aware patch generation,” in *Proc. ICCAD*, 2017. http://cad-contest-2017.el.cycu.edu.tw/Problem_A/default.html

[9] S. Krishnaswamy, H. Ren, N. Modi, and R. Puri, “DeltaSyn: An efficient logic difference optimizer for ECO synthesis,” in *Proc. ICCAD*, 2009

[10] N.-Z. Lee, V. Kravets, and J.-H. Jiang, “Sequential engineering change order under retiming and resynthesis,” in *Proc. ICCAD*, 2017

[11] C.-C. Lin, K.-C. Chen, and M. Marek-Sadowska, “Logic synthesis for engineering change,” *IEEE TCAD*, 1999.

[12] A. Mishchenko, S. Chatterjee, R. Brayton, and N. Een, “Improvements to combinational equivalence checking,” in *Proc. ICCAD*, 2006.

[13] 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,* 2011.

[14] 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*, 2012.

[15] B.-H. Wu, C.-J. Yang, C.-Y. Huang, and J.-H. Jiang, “A robust functional ECO engine by SAT proof minimization and interpolation techniques,” in *Proc. ICCAD*, 2010.

[16] ABC: A system for sequential synthesis and verification. Berkeley Logic Synthesis and Verification Group. http://www-cad.eecs.berkeley.edu/~alanmi/abc

[17] https://www.synopsys.com/dw/ipdir.php?ds=dwc_eco_logic

[18] https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/functional-eco/conformal-eco-designer.html

[19] A. Mishchenko, R. Brayton, A. Petkovska, M. Soeken, L. Amaru, and A. Domic, "Canonical computation without canonical representation,” in *Proc. DAC*, 2018.

[20] https://people.eecs.berkeley.edu/~alanmi/publications/2018/dac18_eco_theorem.pdf

Table 1: Comparison of different algorithms on ICCAD’17 CAD Contest benchmarks

Circuit name	*Circuit information*					*w/o minimize_assumptions*			*w/ minimize_assumptions (the 1st Place in the Contest)*			*SAT_prune+CEGAR_min*		
	#PI	*#PO*	*#gate (F)*	*#gate (S)*	*# target*	*cost*	*# gate (patch)*	*time (sec)*	*cost*	*# gate (patch)*	*time (sec)*	*cost*	*# gate (patch)*	*time (sec)*
unit 1	3	2	6	6	1	15	4	0.05	4	1	0.02	4	1	0.08
unit 2	157	64	1120	1219	1	190	47	0.16	17	4	0.73	17	4	2.81
unit 3	411	128	2074	1929	1	90	3	0.02	80	3	0.05	80	3	4.22
unit 4	11	6	75	77	1	145	2	0.02	42	5	0.03	32	2	0.05
unit 5	450	282	24357	21056	2	3208	40	4.84	47	30	14.20	47	46	86.99
unit 6	99	128	13828	11812	2	5660	6605	111.78	5660	6605	110.77	3942	6232	84.80
unit 7	207	24	2944	1721	1	1076	12	1.22	284	2	3.40	284	2	137.68
unit 8	179	64	2513	3337	1	4211	177	0.36	78	4	3.53	78	4	94.89
unit 9	256	245	5849	4657	4	58	35	0.39	50	29	1.02	80	33	31.32
unit 10	32	129	1581	1956	2	135	587	79.15	135	587	36.80	135	573	61.62
unit 11	48	50	2057	2160	8	4142	1063	124.64	4142	1063	328.35	956	368	65.12
unit 12	46	27	13804	821	1	974	10	0.19	104	1	0.38	104	1	33.98
unit 13	25	39	369	426	1	5661	19	0.10	3467	9	0.65	2656	14	484.64
unit 14	17	15	1981	1006	12	5266	45	6.73	95	42	9.63	94	43	21.94
unit 15	198	14	1886	2262	1	215	1	0.32	191	11	1.23	168	5	22.16
unit 16	417	214	2371	9324	2	2234	41	7.09	318	13	15.17	278	15	42.49
unit 17	136	31	2910	2052	8	698	114	1.11	434	79	2.03	828	93	11.75
unit 18	245	100	4860	3881	1	80	6	1.94	18	1	5.14	18	1	129.07
unit 19	99	128	13349	10787	4	501804	7686	236.03	501804	7686	236.12	490182	5633	83.75
unit 20	1874	7105	30876	34002	4	506	38	0.24	136	6	0.60	120	5	217.35
Geomean						1	1	1x	0.26	0.47	2.12x	0.24	0.43	19.31x