Robust Window-based Multi-node Technology-Independent Logic Minimization

Abstract

Multi-node optimization using Boolean relations is a powerful approach for network minimization. The approach has been studied in theory, and so far its superiority over single node optimization techniques has only been conjectured for practical designs. This is due to the highly memory intensive computations involved in the calculation of Boolean relations representing the multi-node optimization flexibility. In this paper, we present an algorithm to perform Boolean relation-based multi-node optimization using a robust, fast and memory efficient algorithm. In particular, we simultaneously optimize two nodes at a time. We report results on large designs, demonstrating the power of our multi-node optimization algorithm. The robustness of our approach arises from the use of a window-based technique for computing these Boolean relations. Secondly, we perform early quantification during the computation, keeping memory utilization low. Finally, we employ smart heuristics for selecting the node pair to be optimized simultaneously. These features allow the approach to scale well and provide good results for large designs. We perform experiments on a set of large benchmarks and compare our algorithm's performance to a recent SAT-based network optimization technique using complete don't cares. On average, we achieve a 15% reduction in literal count across all the large designs, compared to the complete don't care-based method while maintaining small runtimes and low memory usage.

1. Introduction

The optimization of industrial multi-level Boolean networks is traditionally performed using algebraic techniques. The main reason for this is that traditional Boolean techniques such as don't care-based optimization, though more powerful, do not scale well with design size. Don't cares are calculated for a single node, and they can specify all the flexibility for implementing the node function. These don't cares (for a node) are computed using a combination of Satisfiability Don't Cares (SDCs), Observability Don't Cares (ODCs) or External Don't Cares (XDCs). These are described further in [1].

ODCs [2, 3] of a node are a powerful representation of the node's flexibility. However, the minimization of a node with respect to its ODCs can potentially change the ODCs at other nodes in the circuit, resulting in a need to re-compute ODCs for all circuit nodes. A subset of ODCs, termed as Compatible Observability Don't Cares (CODCs) [2] are computed keeping this limitation in mind. If a node n is minimized with respect to its CODCs, then the CODCs of all other circuit nodes are still valid (and therefore do not need to be recomputed). However, in the CODC computation, the order of selecting a node during the CODC computation becomes important. The maximum flexibility that can be obtained at the fanin node i of a node n is a function of the CODCs of the fanins computed prior to i. In both the ODC and CODC approaches, network optimization is performed on one node at a time.

A significant improvement in terms literal count over don't care-based techniques can be obtained by considering multiple nodes at once. The formulation of such an optimization results in a Boolean relation [4], which implicitly represents the flexibility in optimizing all the nodes simultaneously. The flexibility inherent in multi-node optimization cannot be expressed using functions. Table 1 represents a Boolean relation, which, for a single input vector $\{10\}$, can express more than one *allowed* output vector, $\{00,11\}$. On the other hand, no Boolean function can represent the fact that both vectors $\{00,11\}$ are allowed at the outputs.

Inputs	Outputs
00	00
01	01
10	{00,11}
11	10

Table 1: Example of a Boolean Relation

The superiority of a multi-node optimization approach (using Boolean relations) over don't cares has been pointed out in [5, 6]. The reason for this superior optimization flexibility is that in the computation of a node's don't cares, the functions of all the other nodes are *not* allowed to change. This restriction does not apply to the multi-node optimization approach using Boolean relations since they allow the *simultaneous* modification of all nodes being targeted. However, this superior optimization flexibility has a price. The multi-node optimization approach requires that a Boolean relation be solved, which is typically a highly time and memory intensive operation. As a result, not much attention has been devoted to these approaches, although there have been theoretical works which have suggested the superiority of this technique over don't care-based approaches [6]. However there has been no robust, scalable approach which demonstrates the applicability of multi-node optimization techniques to large designs. This problem is addressed by this paper.

We now illustrate the power of a multi-node optimization approach by way of a small example [7]. Consider the network η shown in Figure 1(a), where node V_1's output f implements the Boolean function $x \cdot y$ and node V_2's output g implements $x + y$. Given a network η with primary outputs Z, the ODC of a node y is given by

$$ODC(y) = \prod_{z_i \in PO(\eta)} \overline{(\frac{\partial z_i}{\partial y})}$$

where,

$$\frac{\partial z_i}{\partial y} = z_i|_y \oplus z_i|_{\overline{y}}.$$

Using this equation for the network of Figure 1, we get $ODC(V_1) = ODC(V_2) = \varnothing$. As a result, no optimization is possible using ODCs. However, we can observe that z is equivalent to $x \oplus y$ as shown in Figure 1(b). This optimization can *only* be obtained when V_1 and V_2 are optimized simultaneously. The Boolean relation resulting from such an optimization can represent this flexibility. After minimizing this Boolean relation, nodes V_1 and V_2 can be deleted from the network without compromising the network's functionality.

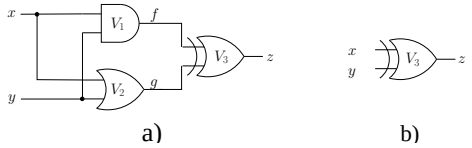

Figure 1: Network η before and after optimization

The main contributions of our work are as follows. We implement a window-based multi-node optimization methodology using Reduced Ordered Binary Decision Diagrams (ROBDDs) [8, 9]. Instead of building the Boolean relation that represents multi-node flexibility in terms of the primary inputs of the network[1] η, we build this relation using a subnetwork η' which is rooted around the nodes being targeted. This allows the resulting Boolean relation to have a *significantly* smaller size, which lets our approach work on large networks. Another feature of our approach that allows it to scale elegantly is that it uses *early quantification* [10] while computing the Boolean relation. Finally, our implementation optimizes two nodes at a time. A careful selection of these node pairs *avoids the quadratic cost* that can result from optimizing all pairs of nodes.

Our multi-node optimization approach results in a Boolean relation that encodes the flexibility in implementing the targeted nodes. To re-implement the targeted functions, this relation needs to be minimized. There are several available techniques to do this [11, 12, 13]; our method uses [11].

The rest of this paper is organized as follows. In Section 2, we provide some preliminaries and definitions. Section 3 contains a summary of previous work in this area, while Section 4 details our approach for multi-node optimization using Boolean relations. In Section 5, we report experimental results and finally, we conclude in Section 6.

2. Preliminaries and Terminology

DEFINITION 1. *A Boolean network η is a directed acyclic graph (DAG) in which every node has a Boolean function f_i associated with it. Also, f_i has a corresponding Boolean variable y_i associated with it, such that $y_i = f_i$.*

There is a directed edge e_{ij} from y_i to y_j if f_j depends explicitly on y_i.

A node y_i is a *fanin* of a node y_j if there is a directed edge e_{ij}. Node y_i is a *fanout* of y_j if there is a directed edge e_{ji}. $FI(y)$ and $FO(y)$ represent the set of immediate fanins and set of immediate fanouts of y respectively.

A node y_i is in the *transitive fanin* of a node y_j if there is a directed path from y_i to y_j. Node y_i is in the *transitive fanout* of node y_j if there is a directed path from y_j to y_i. The transitive fanin of a node y_i up to a k levels, $TFI(y_i, k)$, is the set of nodes $\{y_j\}$ such that there is a directed path of length less than or equal to k, between y_j and y_i. Similarly, the transitive fanout of a node $TFO(y_i, k)$ is the set of nodes $\{y_j\}$ such that there is a directed path of length less than or equal to k, between y_i and y_j.

DEFINITION 2. **Reduced Ordered Binary Decision Diagrams** *are a means to represent a Boolean function f. They are modified Shannon decompositions of f in which any path from the root to the leaves obeys the same variable ordering, and isomorphic nodes are deleted from the decomposition.*

For a given variable ordering, ROBDDs are canonical. In other words, the ROBDDs of equivalent functions are identical.

DEFINITION 3. *The* **consensus operator** *or* **universal quantification** *of a function f with respect to a variable x_i is*

$$\forall_{x_i} f = f_{x_i} \cdot f_{\overline{x_i}}$$

DEFINITION 4. *The* **existential quantification** *of a function f with respect to a variable x_i is*

$$\exists_{x_i} f = f_{x_i} + f_{\overline{x_i}}$$

DEFINITION 5. *A* **Boolean relation** *$\mathcal{R}$ is a one-to-many multi-output Boolean mapping, $\mathcal{R} : B^n \rightarrow B^m$.*

[1] Such an approach would be applicable for only small networks since the ROBDDs of the characteristic function representing the induced relation (on primary inputs) would be intractable for larger networks.

We say that for an output vector $y^l \in B^m$ is *allowed for* an input vector $x^k \in B^n$ iff $(x^k, y^l) \in \mathcal{R}$.

DEFINITION 6. *A* **multi-output Boolean function** *f is a mapping compatible with $\mathcal{R}$ if $f(x) \in \mathcal{R}, \forall_x \in B^n$. This is denoted by $f \prec \mathcal{R}$.*

We refer to the process of finding f given $\mathcal{R}$ as the process of **determinizing** the relation $\mathcal{R}$.

A Boolean relation $\mathcal{R}$ can be represented by its *characteristic function* $\Phi : B^n \times B^m \rightarrow B$ such that $\Phi(x^k, y^l) = 1$ iff $(x^k, y^l) \in \mathcal{R}$.

For a network η which implements the multi-output Boolean function $\boldsymbol{z} = f(\boldsymbol{x})$, the characteristic function is denoted by Φ^η, where

$$\Phi^\eta = \prod_{k=1}^{m} (z_i \overline{\oplus} f_{z_i}(\boldsymbol{x}))$$

where m is the number of outputs of η and $f_{z_i}(\boldsymbol{x})$ is the function of z_i in terms of $\boldsymbol{x}$.

Note that in the sequel a set of variables $\{a\}$ is represented as $\boldsymbol{a}$.

3. Previous Work

Some of the previous research efforts which are relevant to the technique and objective of our paper are discussed next. In [14], the authors describe a method to compute don't cares using overlapping subnetworks, computed using a varying window size. Their method does not optimize wires, but only gates in a design, in contrast to our approach which frequently removes wires in a circuit. Further, this technique uses [15] to optimize a single subnetwork. In [15], optimization is done by manipulating a cover of the subnetwork explicitly. The authors indicate that this requires large amounts of runtime for small networks. As a consequence, the technique of [14], in many examples, requires run-times which are dramatically larger than MIS [16]. The approach of [17] partitions the circuit into subnetworks, each of which is flattened and optimized using ESPRESSO [18].

In [19] a SAT sweeping technique is presented, which identifies and merges functionally equivalent nodes through the use of SAT queries. We run SAT sweep as a preprocessing step to all of our experiments presented in this paper.

In [20], the CODC computation of [21] was shown to be dependent on the current implementation of a node, and an implementation-independent computation was proposed. In [22], the authors perform CODC computation on overlapping subnetworks, and demonstrate a faster technique compared to full CODC computation. They report achieving a good literal count reduction (within 10% of the *full_simplify* (FS) command of SIS [23]) with a faster runtime (25x faster than FS). Our method can achieve a literal count reduction which on average surpasses those reported in [22], within 55% of the runtime of [22]. This improvement is in part due to the additional flexibility encoded in the multi-node optimization technique (using Boolean relations) we present, over the node-by-node optimization in [22].

In [24], the authors present a SAT-based methodology for computing the ODC and SDC, termed as complete DC (CDC), for a every node in a network. They also propose a windowing scheme to maintain robustness. In our experiments, we compare our results with [24], because their approach provided the best literal count reduction and run times of all previous approaches mentioned here.

While the methods discussed above explore the flexibility of exactly one node at a time, a much greater flexibility can be availed by optimizing multiple nodes of a network simultaneously. This is a relatively unexplored aspect of multi-level optimization. There are research efforts which recognize the power of such a technique [5, 25, 6] with respect to multi-node network optimization, but none of these work on even medium sized circuits. The survey described in [5] only points out the advantage of multi-node minimization over don't cares. The approach in [25] describes a BDD-based computation of SPFDs [26], which en-

code the flexibility of more than one node. The average literal count reductions we obtain are better than those reported in [25], even though the size of circuits they report are significantly smaller. Further, their runtimes are much higher and the approach is not shown to be scalable.

In [6], an approach for computing the Boolean relation of a single subnetwork of the original network is described. Our approach, in contrast, demonstrates good results in a scalable manner due in part to the use of an efficient method to find pairs of nodes to optimize together. This method effectively filters out pairs of nodes for which the expected flexibility is low. Also, the results reported in [6] are for very small circuits, and incur extremely high runtimes. Our implementation is powerful and robust, resulting in the ability to optimize large networks extremely fast, with an impressive quality of results. Further, [6] does not use a relation minimizer, but instead it exhaustively calls ESPRESSO in order to minimize the Boolean relation that represents the optimization flexibility. They do acknowledge this as a possible limitation in their paper. We use BREL [11] to minimize the Boolean relations which we construct for each pair of nodes being optimized simultaneously.

There are some earlier research efforts in the context of multi-node optimization using Boolean relations to express the multi-node optimization flexibility, but our approach is very different. A technique which calculates this Boolean relation in terms of primary inputs is presented in [7]. We compute this Boolean relation in terms of only the 'primary input' variables of the extracted subnetwork, enabling the approach to scale elegantly.

A technique to compute the maximal Boolean relation that represents the optimization flexibility for the nodes in an arbitrary subnetwork is presented in [27], which was improved by [28] to additionally compute approximate Boolean relations. However, they do not support their work with experimental results.

The technique for minimizing a Boolean relation used in this paper is reported in [11]. Because the relation will have only two outputs (since we optimize node pairs simultaneously), we use the *QuickSolver* algorithm of BREL to find a solution. This determinizes the first output using the flexibility provided by the relation, and then determinizes the second output, using the constraints imposed by the determinization of the first output. In each of the 2 determinization steps, the cost function for *QuickSolver* is literal count. In the sequel BREL refers only to the *QuickSolver* portion of the algorithm.

4. Our Approach

In general, the exact computation of the Boolean relation expressing the optimization flexibility of multiple nodes of even a small to medium sized network is extremely memory intensive. This is one of the reasons why past research efforts in this area have been mostly theoretical in nature. Our approach for simultaneous multi-node minimization of a multi-level network has several salient features.

- We compute the flexibility for simultaneously optimizing a pair of nodes of the network at a time, using an ROBDD-based approach.
- We avoid memory explosion by a 'windowing' technique which first creates a subnetwork around the two nodes being optimized. This subnetwork has a user-controllable topological depth. The Boolean relation representing the flexibility for simultaneously optimizing the two nodes is built in terms of the primary inputs of the *subnetwork*. This keeps the sizes of the ROBDDs under control, and effectively allows the approach to scale robustly for large networks, with good result quality.
- During the computation of the ROBDD of the characteristic function of the relation, we aggressively control memory utilization by performing careful early quantification.
- Further, instead of running our algorithm on all pairs of nodes, we run the algorithm on only those node pairs that are likely to yield good optimization opportunities. This is done *without* enumerating all node pairs.

Our algorithm begins by efficiently selecting pairs of nodes to optimize from the original multilevel network η. Given a pair of nodes (n_i, n_j) to optimize simultaneously, our algorithm then finds a subnetwork $\eta_{i,j}$ which is rooted around these nodes. The Boolean relation $\mathcal{R}$ representing the simultaneous flexibility of these 2 nodes is computed in terms of the primary inputs of the subnetwork $\eta_{i,j}$. Finally, the Boolean relation $\mathcal{R}$ is minimized using a relation minimizer. The relation minimizer returns a multi-output function (in particular a 2 output function) f, such that f is compatible with $\mathcal{R}$ ($f \prec \mathcal{R}$). The optimized pair of nodes are then grafted back into η. At the end of the for loop we obtain a *minimized* multi-level network $\eta^{'}$.

Algorithm 1 describes the flow of our multi-level optimization methodology. The details of the steps of the algorithm are described in the next subsection.

Algorithm 1 Boolean Relation-based Multi-Node Optimization

$L = select\_nodes(thresh, k_1, k_2, \alpha)$
for all $(n_i, n_j) \in L$ **do**
  $\eta_{i,j} = extract\_subnetwork(n_i, n_j, k_1)$
  $\mathcal{R}^Y(\boldsymbol{s}, \boldsymbol{y}) = build\_relation\_bdd(\eta_{i,j}, X, Z, S, Y)$
  $(n_i^{'}, n_j^{'}) = BREL(\mathcal{R}^Y(\boldsymbol{s}, \boldsymbol{y}))$
  Graft $(n_i^{'}, n_j^{'})$ *in* η
  Delete n_i *and* n_j from η
end for
Return $\eta^{'} = network\_sweep(\eta)$

4.1 Algorithm details

Our Boolean relation-based multi-node network optimization algorithm is shown in Algorithm 1. The input to the algorithm is a Boolean network η, the output is an optimized Boolean network $\eta^{'}$, which is functionally equivalent to η. The primary inputs of η are referred to as X^{global} and its primary outputs are called Z^{global}.

4.1.1 Selecting Node Pairs

When selecting node pairs, we would like to find nodes that share common fanins and fanouts when the subnetwork is created. Not only will this make the subnetwork smaller, but it will also increase the likelihood that don't cares will be found from the resulting relation.

To generate a list of all node pairs to minimize, we call $select\_nodes(thresh, k_1, k_2, \alpha)$. This function starts by selecting a node n_i in the network. To find a potential partner n_j for this node, we first call $TFI_{edge}(n_i, k_1)$, which returns only the nodes $\{m\}$ in the transitive fanin of n_i which have a backward depth of exactly k_1 levels from n_i. From these nodes we call $TFO(m, k_2)$, which returns nodes $\{n_j\}$ in the transitive fanout of m that have a forward depth of up to k_2 levels from m. This gives us all potential partners $\{n_j\}$, and ensures that n_i and n_j will later share at least one common primary input in the subnetwork.

Next, we test each node n_j against n_i to check if the transitive fanins (fanouts) of backward (forward) depth exactly k_1 of both nodes have a support overlap of greater than or equal to *thresh*. We weigh the fanin support overlap by α and the fanout support overlap by 1 - α. We define $TFO_{edge}(n_i, k_1)$ as the nodes in the transitive fanout of n_i which have a forward depth of exactly k_1 levels from n_i. In other words, we test whether

$$\alpha \cdot \frac{|TFI_{edge}(n_i, k_1) \cap TFI_{edge}(n_j, k_1)|}{|TFI_{edge}(n_i, k_1) \cup TFI_{edge}(n_j, k_1)|} + (1-\alpha) \frac{|TFO_{edge}(n_i, k_1) \cap TFO_{edge}(n_j, k_1)|}{|TFO_{edge}(n_i, k_1) \cup TFO_{edge}(n_j, k_1)|} \geq thresh$$

is true. All nodes n_j for which the above test evaluates to be $true$ are paired with n_i and placed in the node pair list L.

These steps are performed for all $n_i \in \eta$, visited in topological order from the POs to the PIs, until every node has been tested for potential partners. A list L of all node pairs to optimize is returned.

We next extract a subnetwork $\eta_{i,j}$ of η, rooted at nodes (n_i, n_j). The technique for this extraction is explained in the following subsection.

4.1.2 Building the Subnetwork

For each pair of node (n_i, n_j) found, we extract sub-networks of η rooted at the nodes n_i and n_j, by calling $extract\_subnetwork(n_i, n_j, k_1)$. This function constructs a subnetwork $\eta_{i,j}$ such that if node $m \in \{TFO(n_i, k_1) \cup TFO(n_j, k_1)\}$, then $m \in \eta_{i,j}$ and if node $p \in \{TFI(n_i, k_1) \cup TFI(n_j, k_1)\}$, then $p \in \eta_{i,j}$. Here k_1 is the same value used when calling $select\_nodes$. The result of this step is illustrated in Figure 2(a) as the shaded subnetwork.

Node $m \in \eta_{i,j}$ is designated as a primary input of $\eta_{i,j}$ if $\exists_{n \in FI(m)}$, $n \notin \eta_{i,j}$. Similarly, a node m is designated as a primary output of $\eta_{i,j}$ if $\exists_{n \in FO(m)}$, $n \notin \eta_{i,j}$. The set of primary inputs (outputs) of $\eta_{i,j}$ is referred to as X (Z).

Next we collect the set of all nodes $m \in TFI(v, k_1)$, where v is a primary output of the subnetwork $\eta_{i,j}$. This step is illustrated in Figure 2(b). Let this set be called D. The nodes in the dotted and shaded region of Figure 2(b) constitute the set D. We then set $\eta_{i,j} \leftarrow \eta_{i,j} \cup D$. Figure 2(c) zooms into the region of interest for the subsequent discussion.

Next, for each $d \in D$ we check if $FI(d)$ can be expressed completely in terms of the current nodes in $\eta_{i,j}$. We perform this check by recursively traversing the network topologically from d towards the primary inputs X^{global} of η. If this traversal visits a node in $\eta_{i,j}$, we terminate the traversal and add all nodes visited in this traversal to $\eta_{i,j}$. If the traversal visits a node in X^{global} instead, then we augment the set of primary inputs of $\eta_{i,j}$ with d, i.e. X is updated as $X \leftarrow X \cup d$. This step is illustrated in Figure 2(d), which shows the final $\eta_{i,j}$. In Figure 2(d), square nodes are the subnetwork PIs X, while triangle nodes are the subnetwork POs Z. Thus, the fanin of the node $w \in D$ is added to $\eta_{i,j}$. Similarly the fanin u of $r \in D$ is next added to $\eta_{i,j}$. This is because both r and w can be expressed completely in terms of X. However, the fanin of node $t \in D$ cannot be expressed in terms of nodes in $\eta_{i,j}$, and so t is added to X. This check avoids the addition of unnecessary primary inputs for representing the sub-network $\eta_{i,j}$. A larger number of primary input variables typically results in larger intermediate ROBDDs in the computation of the Boolean relation $\mathcal{R}$.

Note that the size of each subcircuit $\eta_{i,j}$ is determined by the depth parameter k_1. Hence, by suitably choosing k_1, we ensure that the subcircuits are never too large, and the Boolean relation can be computed with low memory utilization, even for an extremely large network η. The final subnetwork $\eta_{i,j}$ is shown in Figure 2(d). This subnetwork is then used to create a Boolean relation which inherently represents the simultaneous flexibility of both n_i and n_j as discussed in the following subsection.

4.1.3 Computing the Boolean Relation $\mathcal{R}^Y$

As mentioned previously, the exact computation of a Boolean relation expressing the flexibility in a medium to large design could be extremely memory intensive. Also, we employ an ROBDD-based computation for this relation. ROBDDs can, by nature, exhibit very irregular memory requirements, especially for medium to large designs. As a consequence, we have implemented a robust methodology for computing the Boolean relation. Not only do we compute this relation for a node pair (n_i, n_j) using a windowed subnetwork $\eta_{i,j}$ (thus ensuring that the ROBDDs are small) but we also perform careful early quantification and impose limits on the window size to ensure that ROBDDs stay tractable during the relation computation.

Suppose we are given a subnetwork $\eta_{i,j}$, its set of primary inputs X and its set of primary outputs Z. Let the set of nodes being simultaneously optimized be referred to as Y and their combined support be S. Note that S, Y, X and Z correspond to a set of nodes of $\eta_{i,j}$. Let the variables for these be $\boldsymbol{s}$, $\boldsymbol{y}$, $\boldsymbol{x}$ and $\boldsymbol{z}$ respectively as shown in Figure 2(d). The characteristic function of the Boolean relation $\mathcal{R}$ is a mapping $B^{|S|} \times B^{|Y|} \rightarrow B$ s.t.

$$\mathcal{R}^Y(\boldsymbol{s}, \boldsymbol{y}) = \forall_{\boldsymbol{x}}[(\boldsymbol{s} = g_S(\boldsymbol{x})) \Rightarrow \forall_{\boldsymbol{z}}[(\boldsymbol{z} = Z^M(\boldsymbol{x}, \boldsymbol{y})) \Rightarrow \Phi(\boldsymbol{x}, \boldsymbol{z})]]$$

In this expression, $\Phi(\boldsymbol{x}, \boldsymbol{z})$ is the characteristic function of the circuit outputs $\boldsymbol{z} = f(\boldsymbol{x})$. The subexpression $Z^M(\boldsymbol{x}, \boldsymbol{y})$ represents the characteristic function of the circuit outputs expressed in terms of $\boldsymbol{x}$ and $\boldsymbol{y}$. Also, $g_S(\boldsymbol{x})$ is the characteristic function of the $\boldsymbol{s}$ variables in terms of $\boldsymbol{x}$. The computation of $\mathcal{R}^Y$ is explained intuitively as follows. For all primary input minterms $\boldsymbol{x}$, let $\boldsymbol{s}$ take on values dictated by $\boldsymbol{x}$ (i.e. $\boldsymbol{s} = g_S(\boldsymbol{x})$). If so, we should have the situation that if $\boldsymbol{z}$ takes on the values dictated by $\boldsymbol{x}$ and the node values of $\boldsymbol{y}$, then the values of $\boldsymbol{x}$ and $\boldsymbol{z}$ should be related by the original network functionality (i.e. $\Phi(\boldsymbol{x}, \boldsymbol{z})$).

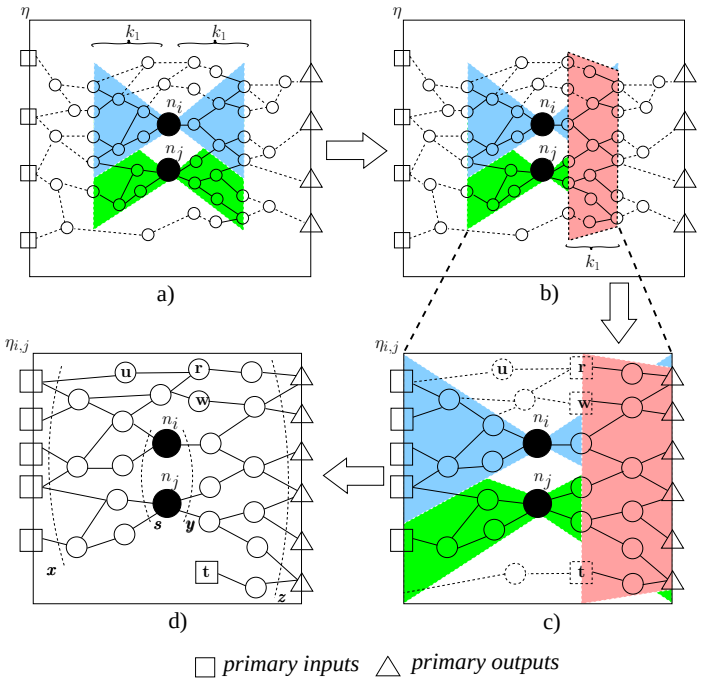

Figure 2: Extraction of Subnetwork

4.1.4 Quantification Scheduling

In our approach, the Boolean relation $\mathcal{R}^Y(\boldsymbol{s}, \boldsymbol{y})$ is computed using ROBDDs. In order to avoid a possible memory explosion problem, we perform early quantification as explained next.

We rewrite the computation for $\mathcal{R}^Y(\boldsymbol{s}, \boldsymbol{y})$ as

$$\mathcal{R}^Y(\boldsymbol{s}, \boldsymbol{y}) = \forall_{\boldsymbol{x}}[(\boldsymbol{s} = g_S(\boldsymbol{x})) \Rightarrow \forall_{\boldsymbol{z}}[\prod_i (z_i \overline{\oplus} Z_i^M(\boldsymbol{x}, \boldsymbol{y})) \Rightarrow \prod_i (z_i \overline{\oplus} Z_i(\boldsymbol{x}))]]$$

This expression can be re-written as:

$$\mathcal{R}^Y(\boldsymbol{s}, \boldsymbol{y}) = \forall_{\boldsymbol{x}}[(\boldsymbol{s} = g_S(\boldsymbol{x})) \Rightarrow \forall_{\boldsymbol{z}}[\prod_i [(z_i \overline{\oplus} Z_i^M(\boldsymbol{x}, \boldsymbol{y})) \Rightarrow (z_i \overline{\oplus} Z_i(\boldsymbol{x}))]]]$$

Our first observation is that the quantification over $\boldsymbol{z}$ ($\forall_{\boldsymbol{z}}$) and the product term over i ($\prod_i$) can be swapped to obtain a new expression

for $\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y})$:

$$\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y}) = \forall_{\boldsymbol{x}}[(\boldsymbol{s} = g_S(\boldsymbol{x})) \Rightarrow \prod_i [\forall_{\boldsymbol{z}}[(z_i \overline{\oplus} Z_i^M(\boldsymbol{x},\boldsymbol{y})) \Rightarrow (z_i \overline{\oplus} Z_i(\boldsymbol{x}))]]]$$

This is correct because in general,

$$\forall_\omega(f \cdot g) = \forall_\omega(f) \cdot \forall_\omega(g)$$

Quantifying out the $\boldsymbol{z}$ variables earlier results in smaller intermediate ROBDDs for the expression to the right of the first implication. The computation can therefore be expressed as:

$$\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y}) = \forall_{\boldsymbol{x}}[(\boldsymbol{s} = g_S(\boldsymbol{x})) \Rightarrow P(\boldsymbol{x},\boldsymbol{y})] = \forall_{\boldsymbol{x}}[\overline{(\boldsymbol{s} = g_S(\boldsymbol{x}))} + P(\boldsymbol{x},\boldsymbol{y})]$$

where P(**x**,**y**) is is the ROBDD obtained after applying the first observation.

$$P(\boldsymbol{x},\boldsymbol{y}) = \prod_i [\forall_{\boldsymbol{z}}[(z_i \overline{\oplus} Z_i^M(\boldsymbol{x},\boldsymbol{y})) \Rightarrow (z_i \overline{\oplus} Z_i(\boldsymbol{x}))]]$$

In general, however,

$$\forall_\omega(f+g) \neq \forall_\omega(f) + \forall_\omega(g)$$

Let the common variables between f and g be ω^*. Let $\omega' = \omega \cap \omega^*$. Then,

$$\forall_\omega(f+g) = \forall_{\omega'}(\forall_{\omega \setminus \omega'}(f) + \forall_{\omega \setminus \omega'}(g))$$

Our second observation is that $g_S(\boldsymbol{x})$ depends on a smaller subset ($\boldsymbol{x'}$) of the primary inputs ($\boldsymbol{x}$) of the network. Hence, we can compute $\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y})$ as

$$\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y}) = \forall_{\boldsymbol{x'}}[\forall_{\boldsymbol{x} \setminus \boldsymbol{x'}} \overline{(\boldsymbol{s} = g_S(\boldsymbol{x}))} + \forall_{\boldsymbol{x} \setminus \boldsymbol{x'}}(P(\boldsymbol{x},\boldsymbol{y}))]$$

which reduces to:

$$\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y}) = \forall_{\boldsymbol{x'}}[\overline{(\boldsymbol{s} = g_S(\boldsymbol{x}))} + \forall_{\boldsymbol{x} \setminus \boldsymbol{x'}}(P(\boldsymbol{x},\boldsymbol{y}))]$$

In practice, we apply both observations in tandem. We first find $g_S(\boldsymbol{x})$, and the set $\boldsymbol{x'}$. Then, while computing P(**x**,**y**), we quantify out $\boldsymbol{x} \setminus \boldsymbol{x'}$. The final computing step is

$$\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y}) = \forall_{\boldsymbol{x'}}[\overline{(\boldsymbol{s} = g_S(\boldsymbol{x}))} + P'(\boldsymbol{x'},\boldsymbol{y})]$$

where $P'(\boldsymbol{x'},\boldsymbol{y}) = \forall_{\boldsymbol{x} \setminus \boldsymbol{x'}}(P(\boldsymbol{x},\boldsymbol{y}))$. By implementing both these ideas, we ensure that intermediate ROBDD never blows up in size. The final ROBDD representing $\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y})$ is returned to the calling function.

4.1.5 Endgame

Next, we invoke a call to BREL to minimize $\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y})$. The output of BREL is a pair of completely specified functions for the nodes n_i' and n_j' such that these functions are compatible with $\mathcal{R}^Y(\boldsymbol{s},\boldsymbol{y})$ and the total cost of n_i' and n_j' is minimal. We finally graft n_i' and n_j' back into η and delete the original nodes n_i and n_j from η.

At the end of the *for* loop, when all node pairs have been processed by our relation-based minimization procedure, we run the *network_sweep* command of SIS [23]. We also verify that the resulting network η' is functionally equivalent to the original network η.

Additionally, for all nodes that were *not* modified by our two-node relation approach, we attempted to find the single node ODCs and minimize them as a post-processing step. The resulting improvement was negligible, and thus we do not include this step in the following results.

5. Experimental Results

The preliminary implementation of our approach was done in SIS [23]. We used large circuits from the *mcnc91* and *itc99* benchmarks for our experiments. The ROBDD package used was the CUDD package [9]. We compare our results against the SAT-based CDC method *mfsw* [24] included in MVSIS [29], which uses a 2x2 windowing technique. Note that all results reported in this section are the literal counts measured by summing the factored form literal count of all nodes in the network. The experiments were performed on a Linux-based Dell Optiplex with a 2.6GHz Core 2 Quad CPU with 4 GB of RAM.

The values for the parameters passed to $select\_nodes()$ are $thresh = 0.8, k_1 = 1, k_2 = 1$ and $\alpha = 0.3$. These values were chosen by varying each parameter according to Table 2 and calculating the average literal count and runtime over the set of 15 circuits. The parameter α was varied first, setting the other parameters to their nominal values. This is because α chooses whether PI or PO compatibility is weighed more, which is independent of the other parameters. Once α was determined, the other parameters determining window size and selectivity were then found using the same method. Various plots describing the runtime and literal count reduction as a function of the swept parameters are omitted due to lack of space. It was found that a 1x1 window yielded a large literal count reduction in addition to keeping runtimes low. The final values for all parameters were found to work equally well across all circuits, and they used in the experiments reported in Table 3 and Table 4. In addition, we impose limitations on the subnetwork size. In particular, if a subnetwork has more than 50 inputs or outputs, then the node pair is rejected without calculating $\mathcal{R}^Y$. This assures that the BDD sizes remain tractable during the computation of the Boolean relation.

Parameter	Low	High	Increment	Nominal
α	0	1.0	0.1	-
k_1	1	3	1	2
k_2	1	4	1	3
$thresh$	0	1.0	0.1	0.5

Table 2: Initial Values, Final Values, Increments, and Nominal Values of the Node Selection Parameters

For Table 3, we first run SAT sweep [19] to remove functionally equivalent nodes from the network. This is the starting point for both the *mfsw* technique and our method, and was similarly performed as a preprocessing step in [24]. The literal count after SAT sweep is shown in Column 2 of Table 3. The literal count after running *mfsw* and our method is reported in Column 3 and Column 4, respectively. These literal counts are measured by summing the factored form literal count of each node across the entire network. Column 5 shows the ratio of literals in Column 4 to Column 3. Column 6 is the peak number of ROBDD nodes in memory for our method. All runtimes are under 90 seconds, which shows that this method scales well with very large circuits.

We can see from Table 3 that starting from the SAT swept circuits, our method reduces the literal count by 15% *over and above* what *mfsw* can achieve. The memory requirements are also very low regardless of the size of the circuit. In [24] it was reported that running SAT sweep and then *mfsw* decreased the literal count by 10%. In Table 3, we report the literal count after SAT sweep (Column 2) and after SAT sweep and *mfsw* (Column 3). Note that the gain due to *mfsw* over SAT sweep is on average 2% based on Table 3.

In Table 4, we start with the reduced network produced by SAT sweep and *mfsw* combined. This table is meant to test if our method can improve on the result obtained by running SAT sweep followed by *mfsw*. Column 2 reports the literal count after running SAT sweep followed *mfsw*. Column 3 shows the literal count after we run our relation method on the netlist obtained by running SAT sweep followed by *mfsw*. We find that an additional 16% literal reduction can be acheived by our technique as shown in Column 4. Also, Column 5 demonstrates that our memory utilization is very low.

It is also important to note that the window size of *mfsw* is the only parameter which can be changed to trade off runtime and literal count improvement. By choosing a 15x15 window for *mfsw*, the runtime matches that of our approach; however, the literal count of *mfsw* improves by less than 0.5% across all the circuits (compared to the results for *mfsw* with a 2x2 window).

	literals				
circuit	sat sweep	mfsw	relation	ratio	memory
c1355	992	992	598	0.603	170
c1908	759	748	605	0.809	293
c2670	1252	1197	933	0.779	320
c5315	3062	2935	2343	0.798	804
c7552	3796	3549	2994	0.844	641
b15	15084	14894	14205	0.954	1210
b17	49096	48595	45892	0.944	1342
b20	22037	21816	19810	0.908	666
b21	22552	22306	20341	0.912	597
b22	33330	33001	30133	0.913	846
s1494	1239	1177	1171	0.995	335
s5378	2327	2283	1978	0.866	617
s13207	5052	4833	4045	0.837	132
s15850	6624	6342	5217	0.823	153
s38417	17531	17314	14719	0.850	619
average	-	-	-	0.850	-

Table 3: Results after SAT Sweep

	literals			
circuit	sat sweep + mfsw	relation	ratio	memory
c1355	992	598	0.603	189
c1908	748	601	0.803	29
c2670	1197	926	0.774	333
c5315	2935	2264	0.771	632
c7552	3549	2813	0.793	404
b15	14894	14085	0.946	605
b17	48595	45444	0.935	1093
b20	21816	19662	0.901	580
b21	22306	20129	0.902	518
b22	33001	29894	0.906	828
s1494	1177	1139	0.968	179
s5378	2283	1956	0.857	628
s13207	4833	4045	0.837	132
s15850	6342	4958	0.782	166
s38417	17314	14539	0.840	619
average	-	-	0.836	-

Table 4: Results after SAT Sweep and *mfsw*

Table 5 shows some properties of the node pairs selected by our algorithm. The values reported in Table 5 are averaged across all the designs we report in Tables 3 and 4. Because of the low value of α chosen, the number of fanouts in common is higher than the number of fanins in common. In addition, due to the 1x1 window chosen, there are very few nodes in the subnetwork and relation, which results in small runtimes. It should be noted that with the high threshold value chosen, all node pairs processed reduced the literal count by at least 1 literal .

6. Conclusions

In this paper, we present an algorithm to perform multi-node optimization using Boolean relations. Our algorithm is robust, scalable and memory efficient. We use an ROBDD-based approach for computing the Boolean relations that express multi-node optimization flexibility. Using a window-based technique for computing these Boolean relations (for simultaneously optimizing two nodes at a time) allows our approach to scale elegantly. We perform early quantification to control the ROBDD size at all stages of the Boolean relation computation, and also employ smart heuristics for selecting the node pairs to be optimized together.

Our results demonstrate the efficiency and scalability of our algorithm. We achieve a better literal count than [24] across all examples, and can improve upon the results generated in [24] by 15%. Runtimes for our method are all under 90 seconds, even for the largest examples, and the memory utilization is very low.

Property	Average
Total number of fanins of both nodes	4.36
Total number of fanouts of both nodes	2.03
% of fanin in common between both nodes	33%
% of fanout in common between both nodes	50%
Total number of literals in both nodes	4.42
Total number of cubes in both nodes	3.23
Total number of PIs of subnetwork	7.31
Total number of POs of subnetwork	4.32
% of subnetwork PIs in common between both nodes	35%
% of subnetwork POs in common between both nodes	50%
Number of nodes in subnetwork	6.97
Number of literals in subnetwork	10.60
Number of BDD nodes in relation	7.13

Table 5: Properties of Node Pairs

References

[1] S. Hassoun, ed., *Logic Synthesis and Verification*. Kluwer Academic Publishers, Nov 2001.

[2] H. Savoj and R. K. Brayton, "The use of observability and external don't cares for the simplification of multi-level networks," in *Proceedings, Design Automation Conference*, pp. 297–301, June 1990.

[3] Y. Jiang and R. K. Brayton, "Don't cares and multi-valued logic network minimization," in *Proceedings, International Conference on Computer Aided Design*, Nov 2000.

[4] R. K. Brayton and F. Somenzi, "Boolean relations and the incomplete specification of logic networks," in *Proceedings, International Conference on Very Large Scale Integration*, Aug 1989.

[5] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," in *Proceedings of IEEE*, pp. 264–300, Feb 1990.

[6] B. Wurth and N. Wehn, "Efficient calculation of boolean relations for multi-level logic optimization," in *Proceedings, European Design and Test Conference*, pp. 630 – 634, Feb 1994.

[7] K. C. Chen and M. Fujita, "Efficient sum-to-one subsets algorithm for logic optimization," in *Proceedings, Design Automation Conference*, pp. 443–448, 1992.

[8] R. E. Bryant, "Graph based algorithms for Boolean function representation," *IEEE Transactions on Computers*, vol. C-35, pp. 677–690, August 1986.

[9] F. Somenzi, "CUDD: CU decision diagram package." http://vlsi.colorado.edu/∼fabio/CUDD/cuddIntro.html.

[10] J. R. Burch, E. M. Clarke, and D. E. Long, "Symbolic model checking with partitioned transition relations," in *Proceeding of International Conference on Very Large Scale Integration*, August 1991.

[11] D. Baneres, J. Cortadella, and M. Kishinevsky, "A recursive paradigm to solve Boolean relations," in *Proceedings, Design Automation Conference*, pp. 416–421, 2004.

[12] Y. Watanabe and R. Brayton, "Heuristic minimization of multi-valued relations," in *Proceedings, IEEE Transactions on Computer-Aided Design*, pp. 1458–1472, Oct 1993.

[13] F. Somenzi and R. K. Brayton, "An exact minimizer for Boolean relations," in *Proceedings, IEEE International Conference on Computer Aided Design*, pp. 316–319, Nov 1989.

[14] J. C. Limqueco and S. Muroga, "Optimizing large networks by repeated local optimization using windowing scheme," in *IEEE International Symposium on Circuits and Systems, ISCAS*, vol. 4, pp. 1993–1996, May 1992.

[15] J. C. Limqueco and S. Muroga, "SYLON-REDUCE: An MOS network optimization algorithms using permissible functions," in *Proceedings, IEEE International Conference on Computer Design*, pp. 282–285, Sept 1990.

[16] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. on CAD/ICAS*, vol. CAD-6(6), pp. 1062–1082, Nov 1987.

[17] S. Dey, F. Brglez, and G. Kedem, "Circuit partitioning and resynthesis," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 29.4/1 –29.4/5, May 1990.

[18] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[19] A. Kuehlmann, "Dynamic transition relation simplification for bounded property checking," in *Proceedings, IEEE International Conference on Computer Aided Design*, pp. 50–57, 2004.

[20] R. Brayton, "Compatible output don't cares revisited," in *Proceedings, IEEE Transactions on Computer-Aided Design*, pp. 618–623, Nov 2001.

[21] H. Savoj, R. Brayton, and H. Touati, "Extracting local don't cares for network optimization," in *Proceedings, IEEE Transactions on Computer-Aided Design*, pp. 514–517, Nov 1991.

[22] N. Saluja and S. P. Khatri, "A robust algorithm for approximate compatible observability don't care (CODC) computation," in *Proceedings, Design Automation Conference*, pp. 422–427, 2004.

[23] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Tech. Rep. UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[24] A. Mischenko and R. K. Brayton, "SAT-based complete dont care computation for network optimization," in *Proceedings, Design, Automation and Test in Europe*, pp. 412–417, 2005.

[25] S. Sinha and R. K. Brayton, "Implementation and use of SPFDs in optimizing Boolean networks," in *Proceedings, IEEE International Conference on Computer Aided Design*, pp. 103–110, 1998.

[26] S. Yamashita, H. Sawada, and A. Nagoya, "A new method to express functional permissibilities for LUT based FPGAs and its applications," in *Proceedings, IEEE International Conference on Computer Aided Design*, pp. 254–261, Nov 1996.

[27] E. Cerny and M. A. Marin, "An approach to unified methodology of combinational switching circuits," in *Proceedings, IEEE Transactions on Computers*, vol. 26, pp. 745–756, Aug. 1977.

[28] H. Savoj and R. K. Brayton, "Observability relations for multi-output nodes," in *Proceedings, International Workshop on Logic Synthesis*, May 1993.

[29] M. G. et al., "Optimization of multi-valued multi-level networks," in *Proceedings of the International Symposium on Multiple-Valued Logic*, pp. 168–177, 2002.