Disjoint-Support Decomposition and Extraction for Interconnect-Driven Threshold Logic Synthesis

Hao Chen, Shao-Chun Hung, and Jie-Hong R. Jiang
Department of Electrical Engineering / Graduate Institute of Electronics Engineering
National Taiwan University, Taipei 10617, Taiwan

ABSTRACT
Threshold logic circuits are artificial neural networks with their neuron outputs being binarized, thus amenable for efficient, multiplier-free, hardware implementation of machine learning applications. In the reviving threshold logic synthesis, this work lays the foundations of disjoint-support decomposition and extraction operation of threshold logic functions. They lead to a synthesis procedure for interconnect minimization of threshold logic circuits, an important, but not well addressed, objective in both neural network and nanometer circuit designs. Experimental results show that our method can efficiently and effectively reduce interconnect as well as weight/threshold value over highly optimized circuits, thus suitable for implementation using emerging technologies.

CCS CONCEPTS
• Hardware → Combinational synthesis; Circuit optimization;

KEYWORDS
decomposition, extraction, threshold logic

ACM Reference Format:

1 INTRODUCTION
As Moore’s law hits the physical limit, new alternative computing architectures and devices are under active research. On the architecture side, there is non-von Neumann architecture for neuromorphic computing, which can achieve effective implementation of neural networks for machine learning applications. On the device side, alternatives beyond CMOS, such as resonant tunneling devices (RTD) [20], quantum cellular automata [3], single electron transistors [12], memristors [9], and spintronic devices [8], have been demonstrated. Threshold logic turns out to be a promising circuit model under these architecture and device innovations based on neuron-like computation. Various nanoscale devices have been used to implement threshold logic circuits [1]. On the other hand, recent progress in deep learning has made neural networks become a popular model to perform tasks in artificial intelligence. Hardware acceleration of neural networks is an active research area. Neural network binarization for effective hardware realization has been demonstrated [6]. Essentially threshold logic networks are neural networks with their activation functions being binarized. The advancements of synthesis and verification of threshold logic circuits may have important practical implications. These trends revive threshold logic research in recent years.

Among prior efforts on threshold logic synthesis, prior work [19, 20] decomposes Boolean functions into a network of threshold logic functions based on Shannon expansion. A decomposition algorithm based on the truth table and binate splitting heuristic is proposed in [19]. In [10], a tree matching method is applied for threshold circuit synthesis. In [17], an implicit-implicit method is proposed. In [16], the and-inverter graph (AIG) and cut-based technology mapping are applied. In [11], [4], and [13], rewiring, merging and collapse operations are proposed, respectively. Common synthesis objectives of threshold logic circuits are minimizing gate counts, circuit depths, and or the magnitudes of weight/threshold values. In this work, we address the synthesis objective from a different aspect of minimizing the interconnect complexity of threshold logic circuits, which is a crucial issue in machine learning applications as neural networks are often layerwise densely connected. Recently, a synthesis method [5] is proposed to reuse partial summation in binarized neural networks for interconnect minimization. However its detailed implementation assumption of a binarized neuron makes it not applicable to threshold logic synthesis.

In this paper, we lay the theoretical foundation toward disjoint-support decomposition for threshold logic functions and extraction of common sub-threshold logic functions for a given set of multiple threshold logic functions. Essentially the enabling technique is the generalized decision list, which we define as an extension to the decision list [7, 18], for Boolean function representation. Further we seek practical applications of the extraction operation to interconnect minimization of threshold logic circuits. Such minimization may directly benefit area cost reduction for implementations with the spintronics- and memristor-based technologies. Experimental results demonstrate that effective interconnect reduction by 10% (resp. 8%) on average can be achieved efficiently on threshold logic circuits that have been highly optimized for delay (resp. area). As a by-product, the magnitudes of weight/threshold values of threshold logic gates can be reduced by 14% (resp. 13%) on average for delay-minimized (resp. area-minimized) circuits. This effect also benefits area cost reduction in RTD-based technology for threshold logic implementation.

The remaining sections are organized as follows. Section 2 gives the preliminaries. The decomposition and extraction operations are detailed in Sections 3 and 4, respectively. Section 5 shows the experimental results, and Section 6 concludes this paper.

2 PRELIMINARIES
A literal is a variable or the negation of a variable. A cube is a conjunction of literals, and will be alternatively treated as a set of literals in the sequel. Given a Boolean function \( f(x_1, \ldots, x_i, \ldots, x_n) \), its negative cofactor, i.e., \( f(x_1, \ldots, 0, \ldots, x_n) \), and positive cofactor, i.e., \( f(x_1, \ldots, 1, \ldots, x_n) \), on variable \( x_i \) are denoted as \( f_{\neg x_i} \) and \( f_{1x_i} \), respectively. The notion of cofactor on a single literal...
can be extended to a cube. That is, the cofactor of \( f \) on a cube \( c = l_1 \land \cdots \land l_k \) is defined as iterative cofactoring \( f \) on literals \( l_1, \ldots, l_k \) and denoted as \( f|_c \). Given an (ordered) set \( X \) of Boolean variables, its set of valuations/assignments is denoted as \( \{X\} \). E.g., \( \{x_1, x_2\} = \{00, 01, 10, 11\} \). An assignment to a set \( X \) of variables is alternatively represented as a cube. E.g., assignment \( (x_1 = 1, x_2 = 0) \) is represented as \( x_1 \land \neg x_2 \). The onset, denoted \( f^1 \), and offset, denoted \( f^0 \), of a function \( f \) over variables \( X \) are the sets of assignments that evaluate \( f \) to 1 and 0, respectively. A function \( f \) is said positive unate (resp. negative unate) on variable \( x_i \) if the implication \( f|_{\neg x_i} \rightarrow f|_{x_i} \) (resp. \( f|_{\neg x_i} \rightarrow f|_{\neg x_i} \)) holds. A function \( f \) is called a unate function if it is either positive or negative unate on every variable. A function \( f \) is said symmetric on variables \( x_1, x_j \) if \( f(\ldots, x_i, \ldots, x_j, \ldots) = f(\ldots, x_j, \ldots, x_i, \ldots) \). In fact, \( f \) is symmetric on variables \( x_i \) and \( x_j \) if and only if \( f|_{\neg x_i \land \neg x_j} = f|_{x_i \land x_j} \). As the symmetry property is transitive, i.e., if variables \( x_1 \) and \( x_2 \) are symmetric and \( x_2 \) and \( x_3 \) are symmetric, then \( x_1 \) and \( x_3 \) are symmetric in \( f \). In this case, \( \{x_1, x_2, x_3\} \) forms a symmetric group.

A threshold logic function (TLF) [15] \( f : \mathbb{B}^n \rightarrow \mathbb{B} \) over input (support) variables \( x_1, \ldots, x_n \) is a Boolean function that can be specified with a vector of parameters, denoted \([w_{i_1}, \ldots, w_{i_n}; T]\), such that

\[
    f = \begin{cases} 
        1, & \text{if } \sum_{i=1}^{n} w_{i_i} x_i \geq T, \\
        0, & \text{otherwise,}
    \end{cases}
\]

where parameter \( w_{i_i} \in \mathbb{Z} \) represents the weight of input \( x_i \), and \( T \in \mathbb{Z} \) is the threshold value. Note that a TLF \( f \) must be a unate function. Specifically, \( f \) is positive (resp. negative) unate on \( x_i \) if \( w_{i_i} > 0 \) (resp. \( w_{i_i} < 0 \)). Any TLF can be converted to a function that is positive unate on every variable by inverting its negative unate inputs [15]. In the sequel, unless otherwise stated we assume that the weights of a TLF have been transformed to positive values.

A threshold logic network (TLN), or called a threshold logic circuit (TLC), is a directed acyclic graph \( G = (V, E) \), where \( V \) is a set of vertices and \( E \subseteq V \times V \) is a set of edges. The set \( V \) consists of three disjoint subsets: primary inputs (PIs), primary outputs (POs), and intermediate nodes representing threshold logic gates (TLGs), where each TLG is referred to as a physical primitive unit that implements a TLF. The set \( e = (u, v) \in E \) signifies the interconnections from vertex \( u \) to \( v \). In this case, we say \( u \) is a fanin of \( v \), denoted \( u \in FI(v) \), and \( v \) is a fanout of \( u \), denoted \( v \in FO(u) \). For a TLG \( v \) with \( n \) fanins \( v_1, \ldots, v_n \), it realizes some threshold function \( f_v \), specified by \([w_{v_1}, \ldots, w_{v_n}; T_v]\) over variables \( x_{v_1}, \ldots, x_{v_n} \), where \( x_{v_i} \) corresponds to the output variable of \( v_i \).

The implementation cost of a TLN can be measured by the total cost of its constituent TLGs and interconnections. To implement a TLG, several approaches including CMOS and emerging technologies have been proposed. For RTD-based implementations, the area of a TLG is mainly affected by the magnitudes of weight and threshold values. Hence the total area of a TLN is estimated by the summation of weights and threshold value in each TLG; while for spintronics-based and memristor-based implementations, the most critical factor which determines the area of a TLG is the total number of fanins [8, 9, 20].

### 3 Threshold Logic Decomposition

#### 3.1 Generalized Decision List

To characterize the feasibility of the threshold gate decomposition, we extend the concept of decision list [7, 18], and define the generalized decision list.

**Definition 1.** The generalized decision list (GDL) of a function \( f \) over variables \( X = \{x_1, \ldots, x_n\} \) is an ordered list of three-tuple nodes

\[
    (X_1, \alpha_i, \beta_i), \ldots, (X_m, \alpha_m, \beta_m)
\]

where \( X_1, \ldots, X_m \) are non-empty variable sets forming a partition on \( X \), i.e., \( \bigcup_j X_j = X \) and \( X_i \cap X_j = \emptyset \) for \( i \neq j \), and \( \alpha_i \subseteq \{X_j\} \) and \( \beta_i \subseteq \{X_j\} \) are the sets of assignments on \( X_j \), along with any assignment on \( X_j \), not in \( \alpha_i \cup \beta_i \), for \( j = 1, \ldots, i - 1 \), that evaluates \( f \) to 1 and 0, respectively, regardless of the assignments on variables \( X_{i+1} \cup \cdots \cup X_m \).

**Example 1.** The Boolean function \( f = x_1 \lor (x_2 \land x_3) \) can be expressed by the GDL: \([x_1; \{1\}, \emptyset], (x_2, x_3; \{1\}, \{1, 10, 00\})\).

**Algorithm 1 ConstructGDL(f)**

**Input:** A TLF \( f(x_1, \ldots, x_n) \) with \([w_{i_1}, \ldots, w_{i_n}; T]\) for \( w_i \geq w_{i+1} > 0 \).

**Output:** A GDL \( L \) of \( f \).

1: \( L := \emptyset, X := \emptyset; \)
2: for \( i = 1 \) to \( n \)
3: \( X \leftarrow X \cup \{x_i\}; \)
4: if no non-controlling assignment on \( X \)
5: add node (\( X, \text{controlling}(X), \text{controlling}(\neg X) \)) into \( L \);
6: break;
7: else if \( f|_{\emptyset} = f|_b \) for all \( b \in \neg \text{non-controlling}(x_1, \ldots, x_i) \)
8: add node (\( X, \text{controlling}(X), \text{controlling}(\neg X) \)) into \( L \);
9: \( X := \emptyset; \)
10: return \( L \);

For a GDL node \((X_i, \alpha_i, \beta_i)\), an assignment in \( \alpha_i \) (resp. \( \beta_i \)) is referred to as a controlling-1 (resp. controlling-0) assignment, and an assignment not in \( \alpha_i \cup \beta_i \) is referred to as a non-controlling assignment.

In fact, any Boolean function can be represented as a GDL. To characterized the feasibility of decomposition and extraction of threshold logic functions, we use GDL to represent a TLF. Algorithm 1 provides a conversion procedure from a TLF to a GDL. The for-loop in Lines 2-9 iterates over the input variables in a descending order with respect to their weight values. In Line 3, the target input \( x_i \) is added into the temporary variable vector \( X \). If there is no non-controlling assignment in \([X]\), the corresponding GDL node is added to the list \( L \) in Line 5, and then \( L \) is ready to be returned in Line 10 as all onset and offset assignments of \( f \) have already been covered. Otherwise, if there is some assignment \( c \in \{\{x_i\}, \ldots, \{1\}\} \) under which the value of \( f \) remains undetermined, we check whether all such \( f|_c \) are functionally equivalent. If yes, a corresponding GDL node is created in Line 8, and vector \( X \) is emptied in Line 9. Else, the process continues with another iteration.

The time complexity of algorithm ConstructGDL can be analyzed as follows. Given an \( n \)-variable TLF \( f \), the for-loop in Lines 2-9 will be executed at most \( n \) times. For each iteration, the bottleneck occurs at Line 7, where the checking can be reformulated implicitly as satisfiability checking or done explicitly via enumeration through all non-controlling assignments. Despite the expensive computation, algorithm ConstructGDL can be efficient in practical applications. As prior work on threshold logic synthesis mostly restricts the fanin number of a TLG to a low level, e.g., 8 inputs in [16], the runtime overhead for the checking can be low. Moreover, in application benchmarks, the number of non-controlling assignments equals 1 frequently.

To ease our discussion, we define three types of GDL nodes:

**Definition 2.** A node \((X_i, \alpha_i, \beta_i)\) in a GDL that constructed by algorithm ConstructGDL is a type-0 node if \( X_i \) is a singleton set and
With GDL, we study the feasibility of disjoint-support decomposition of a TLF. Formally, a disjoint-support decomposition of a TLF $f$ is feasible if it can be expressed as $f(x_1, \ldots, x_k)$ with the minimum value of $\sum_{T \in L} w_{T}$, where $T$ is a set of nodes in a GDL whose support decomposition for the TLF in (a) corresponding to the three cases stated in Theorem 2, respectively.

Given a TLN, the GDL $L$ constructed by ConstructGDL is the longest. That is, there exists no other GDL $L'$ of $f$ such that $|L'| > |L|$.

3.2 Disjoint-Support Decomposition

With GDL, we study the feasibility of disjoint-support decomposition of a TLF. Formally, a disjoint-support decomposition of a TLF $f$ over variables $X = A \cup B$ with $A \cap B = \emptyset$ is to rewrite $f(X)$ as $h(A, g(B))$ for $h$ and $g$ being TLFs. We call variables $A$ and $B$ being the free set and bound set, respectively. To avoid trivial decomposition, we shall assume $A \neq \emptyset$ and $|B| \geq 2$.

The following theorem relates GDL and the decomposability of a TLF.

Theorem 2. Given a TLF $f(X)$ with $X = A \cup B$ for $A \neq \emptyset$ and $|B| \geq 2$, a disjoint-support decomposition $f(X) = h(A, g(B))$ is feasible if there exists a GDL $L$ such that one of the following three cases holds:

1. The bound set variables $B$ appear in some contiguous type-0 nodes in $L$.
2. The bound set variables $B$ appear in some contiguous type-1 nodes in $L$.
3. The bound set variables $B$ appear in the nodes of some suffix list of $L$.

The theorem can be proved constructively by deriving functions $h$ and $g$ with respect to the three cases of Theorem 2. For a TLF $f$ with $\{w_1, \ldots, w_n; T\}$, assume the bound set variables $B = \{x_k, \ldots, x_l\}$, where $1 \leq k < l \leq n$. In the first case,

$$g = [1, \ldots, |B|],$$

$$h = [w_1, \ldots, w_{k-1}, w_g, w_{l+1}, \ldots, w_n; T],$$

for $w_g = \sum_{j=k}^l w_j$. In the second case,

$$g = [1, \ldots, 1; 1],$$

$$h = [w_1, \ldots, w_{k-1}, w_g, w_{l+1}, \ldots, w_n; T],$$

for $w_g = w_k$. In the third case,

$$g = [w_k, \ldots, w_n; T - \sum_{x_i \in c} w_i],$$

$$h = [w_1, \ldots, w_{k-1}, w_g; T],$$

for $w_g = T - \sum_{x_i \in c} w_i$, where $c$ is the assignment in $\{(x_1, \ldots, x_{k-1})\}$ with the minimum value of $\sum_{x_i \in c} w_i$ as discussed in suffix TLF derivation. It can be verified that the above derivations establish $f(X) = h(A, g(B))$.

Example 3. Figure 1(b), (c) and (d) show three examples of disjoint-support decomposition for the TLF in (a) corresponding to the three cases stated in Theorem 2, respectively.

The GDL $L$ constructed by ConstructGDL exhibits the nice property that if a bound set $B$ appears in some GDL $L'$ as stated in Theorem 2, then $B$ can also be found in $L$. The property is formally stated in Theorem 3.

Theorem 3. In Theorem 2, it suffices to consider only the longest GDL constructed by algorithm ConstructGDL.

By the above theorems, we know that our GDL construction algorithm can generate the longest GDL of a TLF, and that all options of disjoint-support decomposition stated in Theorem 2 can be covered.

4 THRESHOLD LOGIC EXTRACTION

4.1 Problem Formulation

Given a TLN with multiple TLGs having the same disjoint decomposition option, it is possible to extract a common decomposition TLF for logic sharing among the TLGs. Such an extraction operation may reduce interconnect complexity of the TLN. To compute extraction candidates, we define extraction set and extraction pair of a TLF as follows.

Definition 3. An extraction set is a set of nodes in a GDL whose corresponding variable set forms a legal bound set for disjoint-support decomposition. A set with contiguous type-0 (resp. type-1) nodes is called a type-0 extraction set (resp. type-1 extraction set). An extraction pair $(S, v)$ consists of an extraction set $S$ and a corresponding TLG $v$ where $S$ comes from.

Example 4. For the TLN in Figure 3(a), the set $S = \{(x_3, 0, \{0\}), (x_4, 0, \{0\})\}$ is a type-0 extraction set, and $(S, v_1)$ is an extraction pair.

Problem 1 (Threshold Logic Extraction). Given a TLN, the threshold logic extraction problem seeks to find common decomposition TLGs for logic sharing among the TLGs such that the total number of interconnections in the TLN is minimized while the function of the revised TLN remains unchanged.

4.2 Extraction Flow

To achieve a high-quality solution efficiently, we propose an extraction flow shown in Figure 2. In the flow, an input circuit, not necessarily a TLN, is first synthesized into a TLN through existing TL synthesis tools and then the weights and threshold values of the TLGs are minimized. After the pre-processing, the core of our extraction flow includes three main stages: 1) extraction pair initialization, 2) iterative TLG extraction, 3) and TLG post-processing, to be detailed in the following.
4.3 Extraction Pair Initialization

Given a TLN $G = (V, E)$ with all its TLGs $v \in V$ converted in the positive unate form, the set of extraction pairs can be collected from the GDL, generated by algorithm ConstructGDL, of each TLG $v \in V$ by identifying the longest contiguous type-0 and type-1 nodes in the GDL.  

Algorithm 2 sketches the steps.

Algorithm 2 InitExtractionPairs(G)
Input: A TLN $G = (V, E)$ with every TLG $v \in V$ in positive unate form.
Output: A set $P$ of extraction pairs.

1. $P := \emptyset$;
2. for each TLG $v \in V$
   3. for each $S \in ExtractionSets(v)$
      4. $P := P \cup \{(S, v)\}$;
3. return $P$;

Example 5. Consider the TLN in Figure 3(a). Assume variables $x_1$ and $x_2$ are inverted for TLG $v_1$ to be in positive unate form. For the TLN, the extraction pairs $\{(\neg x_1, x_3, x_4, x_5), v_1\}$, $\{(x_3, x_4, x_5), v_2\}$ and $\{(x_3, x_4, x_5), v_3\}$ can be derived.

4.4 Iterative TLG Extraction

Given a TLN $G = (V, E)$, its corresponding set $P$ of extraction pairs, and an upper bound $N_{\text{max}}$ on logic level, we perform iterative extraction to reduce the interconnect complexity of $G$ while maintaining its logic level within $N_{\text{max}}$. The procedure is sketched in Algorithm 3. In Line 1, the extraction pairs $P$ are sorted such that TLGs with larger extraction sets will be extracted first. Because larger extraction sets may potentially contribute to larger common extraction subsets, the procedure ExtractVertex is more likely to achieve greater interconnect reduction through vertex extraction. In each iteration of the while-loop in Lines 2-11, we scan through all extraction pairs in $P$ and check if the extract operation can be applied. For each extraction pair $(S, v) \in P$, if $v$ is on a critical path, the pair $(S, v)$ is removed from $P$ as an extraction on $v$ will increase the logic level of $G$ by one. Otherwise, a new vertex is extracted by ExtractVertex, and $G$ and extraction pairs are updated.

Algorithm 3 ExtractNtk(G, $P$, $N_{\text{max}}$)
Input: A TLN $G = (V, E)$, extraction pairs $P$, and level bound $N_{\text{max}}$.
Output: A TLN $G' = (V', E')$ after extraction.

1. sort the pairs $(S, v) \in P$ in a descending order of $|S|$;
2. $G' := G$;
3. while $P \neq \emptyset$
4. $P' := \emptyset$;
5. for each $(S, v) \in P$
6. if $v \in \text{CriticalPath}(G')$ and $\text{level}(G') \geq N_{\text{max}}$
7. $P := P \setminus \{(S, v)\}$;
8. else
9. $(G', P_E) := \text{ExtractVertex}(G', (S, v), P)$;
10. $P' := P' \cup P_E$;
11. sort the pairs $(S, v) \in P'$ in descending order of $|S|$;
12. $P := P'$;
13. return $G'$;

The function ExtractVertex identifies the largest common extraction subset, i.e., a subset shared by at least two extraction sets, based on the selected extraction set to form a new TLG, and adds corresponding new extraction pairs. Given a TLN $G'$, a target extraction pair $(S, v)$ and a set of extraction pairs $P$, the algorithm ExtractVertex finds the largest common extraction subset $S_{\text{max}}$ shared by at least two extraction sets. By checking all the extraction sets in $P$ having the same type with $S$, it obtains $S_{\text{max}}$ by set intersection. It then creates a new TLG $v'$ in $G'$ based on $S_{\text{max}}$, and updates the TLGs where $v'$ is extracted from according to the derivation described in Section 3.2.

Example 6. Figure 3(a) shows a TLN with three TLGs and their respective GDLs, where the maximum common extraction subset is identified and indicated by the dash round rectangles. In Figure 3(b), a new threshold gate $v_4$ is constructed by applying the ExtractVertex operation. The fanout gates of $v_4$ are rebuilt, and then interconnections...
between \(v_4\) and these gates are done. As a result, the interconnect complexity is reduced from 17 in Figure 3(a) to 13 in Figure 3(c).

4.5 TLG Post-Processing

In this step, we iterate through each TLG and perform the following processing: For a one-input TLG, if the weight is no smaller than the threshold value, it is equivalent to a buffer and can be replaced with a wire. Otherwise, it is equivalent to a constant 0 gate and the circuit can be simplified with constant propagation. For a multiple-input gate, we minimize its weight/threshold values with ILP-based minimization [14].

Example 7. Figure 3(b) and (c) show the effect of post-processing. Because TLG \(v_2\) is equivalent to a buffer and is removed from the TLN. Furthermore, the weight/threshold values of \(v_1\) and \(v_3\) are minimized.

5 EXPERIMENTAL RESULTS

The proposed extraction algorithm was implemented in the ABC environment [2] using the C programming language. All experiments were conducted on a Linux workstation with an Intel Xeon 2.1GHz CPU with 128GB memory. Test cases were selected from ISCAS, MCNC, and FI benchmark suits. We prepared the initial threshold logic circuits using the state-of-the-art threshold logic synthesis approach [16] implemented in ABC. The circuits with less than 200 threshold gates after synthesis of [16] were not included in our experiments.

Gate count and logic level are common metrics in conventional TLN implementation such as LUT-based design. However, gate count is not the dominant factors of TLN area estimation in implementations based on emerging technologies. For example, the sum of total interconnections is the dominant factor of spintronic-based TLN area estimation rather than the total gate count. To evaluate the solution quality of our proposed algorithm in terms of the cost of emerging technologies as mentioned in Section 2, we defined two cost functions for area estimation. For spintronic- and memristor-based TLN implementation, the main factor affecting circuit area is the magnitude of weight and threshold values. Hence we have

\[
C_{\text{wire}} = \sum_v |FI(v)|,
\]

where \(|FI(v)|\) denotes the number of fanins of threshold gate \(v\) in the TLN. For RTD-based implementation, the dominant factor of circuit area is the magnitude of weight and threshold values. Hence we have

\[
C_{\text{RTD}} = \sum_v \sum_{i=1}^{\text{fanin}(v)} \alpha \cdot (|w_{v_i}| + |T_{v_i}|),
\]

where \(\alpha\) (setting to 1 in the experiments) is the unit area of an RTD, \(w_{v_i}\) is the weight of \(i\)th fanin of gate \(v\), and \(T_{v_i}\) is the threshold value of gate \(v\).

Table 1 shows the results of our extraction algorithm applied on circuits synthesized with [16] under delay and area optimization options. Columns 2 and 3 show the numbers of inputs and outputs of the circuits, respectively; Columns 4, 5, 6, and 7 (resp. 13, 14, 15, and 16) report the number of TLGs, number of logic levels, interconnect cost, and RTD implementation cost of the TLNs synthesized by [16] under delay (resp. area) optimization with weights and threshold values minimized by the method in [14]; Columns 8, 9, 10, 11, and 12 (resp. 17, 18, 19, 20, and 21) list the number of TLGs, number of logic levels, interconnect cost, and RTD implementation cost, and CPU time of entire computation, including the GDL construction and extraction operation, performed on TLNs synthesized by [16] under delay (resp. area) optimization.

According to Table 1, for circuits optimized with delay minimization, our algorithm achieved an average of 10% reduction on \(C_{\text{wire}}\) and 14% reduction on \(C_{\text{RTD}}\) in the cost of 18% increase in logic level. On the other hand, for circuits optimized with area minimization, our algorithm achieved an average of 8% reduction on \(C_{\text{wire}}\) and 13% reduction on \(C_{\text{RTD}}\) in the cost of 6% increase in logic level. Notice that there are more significant \(C_{\text{wire}}\) and \(C_{\text{RTD}}\) reductions on circuits optimized for delay compared to those optimized for area. This fact is understandable as area-driven synthesis may have exploited potential logic sharing extensively. On the other hand, the higher percentage of level increase in delay-optimized, compared to area-optimized, circuits is much due to the original small level values; a slight level increase contributes to an amplified percentage increase. Note that although extraction may potentially increase logic level, the reduced interconnect complexity may nullify or alleviate circuit performance degradation. For the run time, all extraction computations were done within 16 seconds (occurred in the case b19) while most instances were solved almost instantly. The experimental results show that the extraction computation can be effective and efficient.

As the extraction operation reduces interconnect complexity in the cost of increasing logic level, it sometimes may be desirable to explore the trade-off between interconnect complexity and logic level. To achieve such a trade-off, we may specify an upper bound of logic level to about 20, the percentage of level increase in delay-optimized, compared to area-optimized, circuits is much due to the original small level values; a slight level increase contributes to an amplified percentage increase. Note that although extraction may potentially increase logic level, the reduced interconnect complexity may nullify or alleviate circuit performance degradation. For the run time, all extraction computations were done within 16 seconds (occurred in the case b19) while most instances were solved almost instantly. The experimental results show that the extraction computation can be effective and efficient.
Figure 4: Trade-off between interconnect/RTD cost and logic level for circuit b15.

6 CONCLUSIONS

This work has laid the foundations of disjoint-support decomposition of threshold logic functions and proposed an extraction algorithm for interconnect minimization of threshold logic networks. Experimental results have demonstrated the efficiency and effectiveness of the proposed method in reducing interconnect/RTD implementation costs and in the trade-off between interconnect complexity and circuit depth. As interconnect costs become a dominating factor to circuit area and delay, the extraction operation may play a key role for threshold logic synthesis.

ACKNOWLEDGMENT

The authors thank Chia-Chih Chi, Siang-Yun Lee, and Nian-Ze Lee for helping the experiments. This work was supported in part by the Ministry of Science and Technology of Taiwan under grants 105-2221-E-002-196-MY3, 105-2923-E-002-016-MY3, and 106-2923-E-002-002-MY3.

REFERENCES