BESWAC: Boosting Exact Synthesis via Wiser SAT Solver Call

Sunan Zou, Jiaxi Zhang, Bizhao Shi, Guojie Luo
School of Computer Science, *Peking University*, Beijing, China
Center for Energy-efficient Computing and Applications, *Peking University*, Beijing, China
{zousunan, jxzhang, bshi, gluo}@pku.edu.cn

***Abstract*—SAT-based exact synthesis is a critical technique in logic synthesis to generate optimal circuits for given Boolean functions. The lengthy trial-and-error process limits its application in on-the-fly logic optimization and optimal netlist library construction. Previous research focuses on reducing the execution time of each trial. However, unnecessary SAT solver calls and varying execution times among encoding methods remained issues. This paper presents BESWAC to boost exact synthesis from the flow level. It leverages initial value prediction, encoding method selection, and an optional early exit to call SAT solvers efficiently and wisely. Moreover, BESWAC can seamlessly integrate existing acceleration methods focusing on individual trials. Experimental results show that BESWAC achieves a 1.79x speedup compared to state-of-the-art exact synthesis flows.**

***Index Terms*—Logic synthesis, Exact synthesis, SAT problem**

I. Introduction

Exact synthesis tries finding a circuit to implement given Boolean functions with minimum resources. Targeted resources can be the circuit's size or depth, and we only consider size in this paper. It has many applications in electronic design automation (EDA), like logic optimization [1], NPN matching [2], and synthesis for emerging technologies [3]. SAT-based exact synthesis [4] applies a trial-and-error flow, as in Figure 1a. Each trial asks: "Is there a netlist N that can implement function f with r resources?" We formulate the question as a SAT problem and send it to a SAT solver. r is initially set to 0 and gradually increased. The flow stops when the SAT solver finds a satisfiable assignment, resulting in an optimal circuit.

Due to the complexity of exact synthesis, the execution time increases double-exponentially as the number of inputs grows. Therefore, optimal netlist libraries for logic optimization are limited to 5-input functions and some 6-input functions [4]. Furthermore, on-the-fly logic optimization methods using exact synthesis are hindered by unpredictable and time-consuming execution [1]. This situation signifies the need to boost the process by reducing execution time, especially for on-the-fly applications.

The critical point is to reduce the largest portion of the total execution time: the SAT-solving process. Previous efforts focus on two primary strategies from a trial-level view: 1) Offer additional information to SAT solvers: Providing additional information to SAT solvers shrinks the search space and speeds up the solving process. This includes informative symmetry-breaking clauses [5], [6] or encoding topology information [4], [7]. 2) Improve SAT solvers: Enhancing SAT solving [8] enables faster completion of each round of questioning.

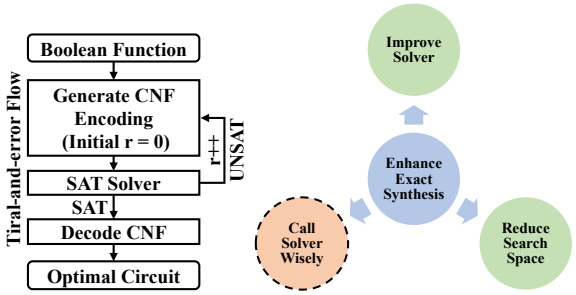


Figure 1: (a) SAT-based exact synthesis flow; (b) Classification of current methods of boosting SAT-based exact synthesis.

However, an untapped optimization area lies in the trial-and-error flow of exact synthesis: **"How to call SAT solver wisely?"** as in Figure 1b. By reducing unnecessary trials and making wiser ones, we can optimize exact synthesis in a flow-level view. Substantial improvements can be achieved by reducing the number of SAT solver calls and selecting the fitting encoding. This new direction targets flow-level optimization and is independent and complementary to the existing ones.

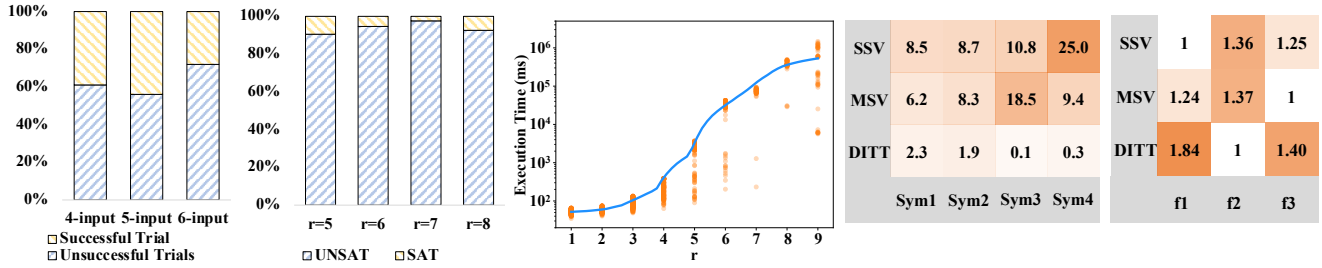
Therefore, we propose BESWAC, a flow-level optimization system for exact synthesis. It incorporates three key techniques: 1) **Initial Value Prediction**: This technique reduces the number of SAT solver calls using circuit lower bounds as the initial value, supported by relevant works and proofs. 2) **Encoding Method Selection**: This technique selects a suitable encoding method for specific Boolean functions. It extracts informative features from the Boolean functions and uses a concise machine learning (ML) model to make informed decisions. 3) **Optional Early Exit**: This strategy allows timely abandonment of particularly challenging SAT solver trials based on empirical rules, thereby saving time with seldom quality loss.

In evaluations on 4-input, 5-input, and 6-input Boolean functions, BESWAC achieved an average 1.79× performance speedup compared to the original exact synthesis flow. The encoding selection technique alone provided a 1.59× performance speedup. While the initial value prediction and optional early exit speedups of 1.11× and 1.34×, respectively.

II. Background and Motivation

SAT-based exact synthesis identifies the optimal netlist to implement a specified Boolean function F. Let $F=(f_1, ..., f_m)$:

979-8-3503-4859-0/DATE24/© 2024 EDAA



(a) (b) (c) (d) (e)

Figure 2: Execution time profiling of exact synthesis: (a) Ratio of successful and unsuccessful trials; (b) Ratio of SAT and UNSAT trials; (c) Trend as r increases; (d) Ratio of best-performance cases achieved by the combinations of encoding (SSV, MSV, and DITT) and symmetry-breaking (Sym1-4) methods. (e) The variance between functions (f1-3) using different encodings.

$\{0,1\}^n \rightarrow \{0,1\}^m$ be a Boolean function with n inputs $(x_1, x_2, ..., x_n)$ and m outputs. Exact synthesis generates a Boolean chain $(x_{n+1}, ..., x_{n+r})$, where $x_i = o_i(x_{j(i)}, x_{k(i)})$ with $j(i) < k(i) < i$, and o_i can implement any 2-input Boolean function. When it is feasible to implement F within r gates, we have $1 \leq l(k) \leq n+r$ for all $1 \leq k \leq m$ such that $f_k = x_{l(k)}$. Figure 1a illustrates the general flow of SAT-based exact synthesis. A Boolean function is encoded into conjunctive normal format (CNF) with initial resource constraint $r=0$. Then, a solving trial loop is executed, and r increments by one unit each round until a satisfiable assignment exists. Finally, decoding the assignment generates the targeted optimal circuit. Alternatively, r can be initialized with an upper bound and gradually decreased [9].

The exact synthesis problem has various encoding methods, including Single Selection Variable (SSV), Multiple Selection Variable (MSV), and Distinct Input Truth Tables (DITT). Additionally, symmetry-breaking methods such as *only non-trivial steps* (N), *use all steps* (A), *no replication of operands* (R), *co-lexicographically ordered steps* (C), *lexicographically ordered operands* (O), and *ordered symmetric variables* (S) can help SAT solvers find conflicting clauses faster. Due to space limitations, we do not delve into the details of these methods in this paper. Interested readers can refer to the summary [4] by Haaswijk et al. Each Boolean function has a suitable encoding method, which we will discuss later.

Exact synthesis flow has several parameters significantly influencing the execution time, like initial value, encoding, and symmetry-breaking methods. Choosing an optimal combination of these parameters is non-trivial. We conducted extensive profiling experiments to understand their effects on execution time. Our investigations aim to determine the computational cost of unsatisfiable trials and the influence of the encoding method. The insights gained will guide efficient and effective SAT-solver calls during exact synthesis. We tested 1,000 randomly selected 4-input and 5-input Boolean functions using the ParKissat solver [8] and various encoding methods.

Figure 2a shows that trials on unsatisfiable r values account for over 50% of the total execution time, even up to 90% in some instances. We also analyzed the execution time trend with increasing r values using MSV encoding on sampled 4-input and 5-input Boolean functions (Figure 2c). SAT solver execution time typically lacks a strong correlation with the number of variables or clauses. However, a consistent trend exists in this specific problem: the growth in execution time slows after taking the logarithm as r increases. These observations indicate that initial value prediction can reduce unsatisfiable trials, saving substantial time. Furthermore, unsatisfiable trials took significantly longer than satisfiable ones (Figure 2b), with the last successful trial usually taking less time than the preceding unsuccessful one. Therefore, it is possible to differentiate challenging UNSAT trials by empirical rules and abandon them early.

We also evaluated the impact of encoding methods using sampled 4-input functions. Figure 2d shows the distribution of Boolean functions performing best with each encoding method. It indicates that optimal encoding varies by function; no single method is universally superior. Encoding method selection based on each Boolean function's unique characteristics has great potential to reduce total execution time. Figure 2e illustrates significant variations in execution time for different encoding methods with the same r, with each cell representing the time relative to the optimal method. The only variable causing such a significant difference in execution time is the Boolean function. The experiment further emphasizes that choosing a suitable encoding method for a specific Boolean function could effectively reduce the exact synthesis execution time.

III. Methods

A. Overview

Based on the observations above, we propose BESWAC to boost exact synthesis from the flow level, as in Figure 3. It encompasses three essential modules: initial value prediction, encoding selection, and optional early exit. These modules boost the original flow (the dotted rectangle) by making wiser decisions when invoking SAT solvers. BESWAC accepts a Boolean function F and computes an initial lower bound r_0 based on F's characteristics, combining general and specific bounds. Then, the encoding selection module extracts F's representative features and leverages a multilayer perceptron (MLP) to determine the suitable encoding method for F. The encoder in BESWAC uses the initial value and the chosen encoding to trigger the original exact synthesis flow. An optional and aggressive early exit module uses empirical heuristics to

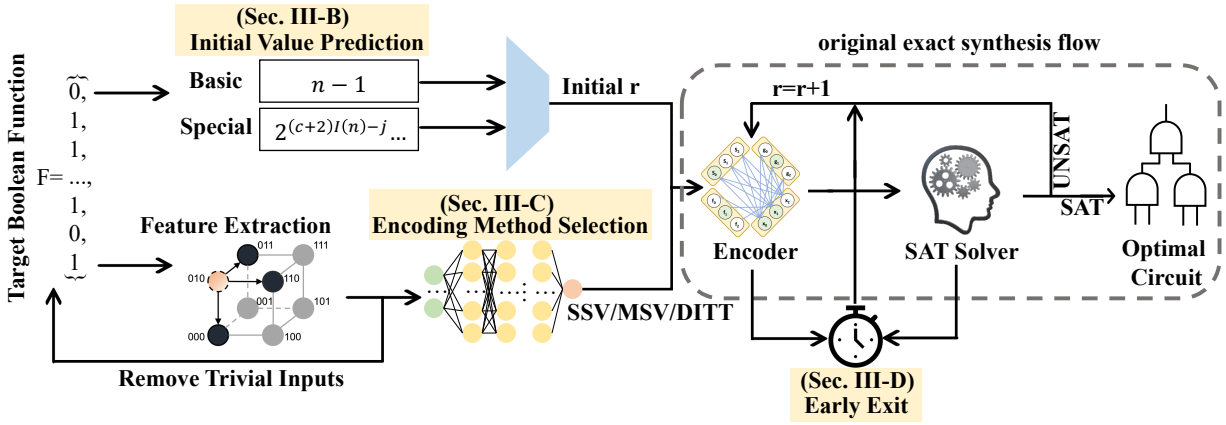


Figure 3: The framework of BESWAC powered by 1) initial value prediction, 2) encoding selection, and 3) optional early exit.

evaluate the hardness of encoding. It abandons identified hard trials early if the execution time exceeds a heuristic threshold. This optional module speeds up exact synthesis at the potential cost of quality, which rarely happens in our experiments.

By accurately determining r_0, BESWAC minimizes trials on infeasible resource values, reducing unnecessary SAT solver calls. The suitable encoding method boosts solver efficiency and speed for all trials. The optional early exit strategy further cuts redundant SAT solver calls, reducing computation time without noticeably impacting result quality.

B. Initial Value Prediction

Exact synthesis targets the realization of a specific Boolean function using minimum resources. During each trial round, the available resource r increases unit by unit. The total execution time depends on the cumulative trials across all rounds. Reducing trials can decrease execution time, as detailed in Section II. Predicting an initial value can help avoid unproductive trials. Circuit lower bounds are well-studied. Most findings focus on circuits under specific constraints, diverging from general exact synthesis [10]–[12]. These application-specific bounds from existing methods are termed special bounds. To adapt general exact synthesis cases, we propose a general basic lower bound, termed the basic bound. We use the basic lower bound as the initial value, supplemented by special bounds for specific functions and constraints.

We classify special bounds into two classes. The first type deals with circuits with only AND, OR, and NOT gates [10]. These constraints help generate an optimal circuit library for And-Inverter Graphs (AIGs) rewrite, a potent logic optimization algorithm [13]. AIG describes a circuit with only AND and NOT gates. Rewrite substitutes subgraphs in AIG with pre-computed ones in the optimal library to reduce the size and depth. The second type is function-specific, like parity or majority [11]. Special bounds often yield more significant results and time savings than basic bounds and are applied prior to basic ones if feasible.

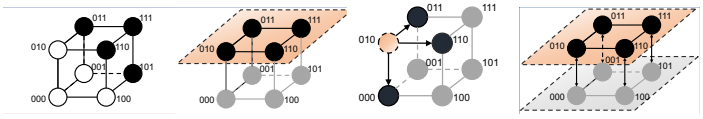
In order to accommodate a broader range of scenarios, we apply a general basic lower bound. Assuming Boolean function $F(x_1, x_2, ..., x_n)$ has n primary inputs and a primary output o_f. We call F non-trivial if each x_i affects the value of o_f and all x_i are not logically equivalent. The primary output o_f of F needs to collect signals from all inputs $(x_1, x_2, ..., x_n)$. The most efficient is using a complete binary tree T, regarding all inputs $(x_1, x_2, ..., x_n)$ as leaf nodes. Any other structure will induce at least one path(s) connecting certain x_i to o_f, and the number of gates will increase if not equal. Furthermore, T has no 1-degree node since we can always shrink a 1-degree node to its parent without reducing the number of leaf nodes. The minimum gates to implement F is no less than the number of non-leaf nodes in T, $\lfloor n/2 \rfloor + \frac{\lceil n/2 \rceil}{2} \times 2 - 1 = n - 1$. Therefore, implementing non-trivial F requires a minimum of $n-1$ gates. This lower bound can potentially reduce overall execution time.

BESWAC analyzes the truth table of the Boolean function and selects the strongest applicable lower bound, r_b, from the available options. We initialize the value r as r_b instead of 0 for the first trial. This approach reduces the number of unsuccessful trials, resulting in a more efficient synthesis process with fewer invocations of the SAT solver. As a result, the execution time is significantly reduced. This boosting technique is nearly overhead-free since these lower bounds can be calculated in constant time complexity. The effectiveness of this technique is validated through experiments in Section IV.

C. Encoding Method Selection

Multiple approaches exist for encoding exact synthesis problems as SAT problems, each with trade-offs regarding variables and clauses. For example, MSV encoding involves fewer variables but more clauses, while SSV encoding follows the opposite pattern. The choice of encoding and symmetry-breaking methods for a given Boolean function can impact execution time greatly, as discussed in Section II. The sole variable in this execution time variation is the Boolean function itself. Thus, selecting the appropriate encoding method for each function is crucial for reducing overall execution time. However, manually classifying numerous functions into the sea of encoding and symmetry-breaking combinations simply by experience alone is impractical.

We propose leveraging machine learning to select the appropriate encoding method for Boolean functions. There are two primary reasons for adopting machine learning in this context: 1) When provided with informative features, machine



(a) (b) (c) (d)

Figure 4: (a) Boolean hypercube of 3-input Majority function; (b) Co-factor (setting input x_2 to 1): the orange face; (c) Sensitivity (input pattern 010): the orange point; (d) Influence of input x_2: the orange and grey face.

learning excels in classification tasks. 2) Ample training data are available for functions with no less than four inputs. We can train a model on a subset of these functions to accelerate the encoding method selection process.

The classification ability of the ML model greatly lies in feature selection. We represent Boolean functions with three informative signatures: co-factor, sensitivity, and influence, as illustrated in Figure 4. These signatures have proven effective in classifying NPN functions [14], a similar Boolean function classification problem. Our feature representation is an $n\times(2^{n-2}+3)$ matrix consisting of an $n\times2$ ordered co-factor vector (OCV), an $n\times2^{n-2}$ ordered sensitivity vector (OSV), and an $n\times1$ ordered influence vector (OIV) for n-input Boolean functions. This matrix captures the face and point characteristics of the functions, treating each function as a hypercube, as in Figure 4a. These characteristics reflect the relationships between output changes and different input patterns. The features OCV, OSV, and OIV are explained below.

Definition 1 (Ordered co-factor vector, OCV)**.** For a Boolean function f with n inputs, an ordered co-factor vector is $OCV(f) = \{|f_{z=v}| : z \in \{x_1, x_2, ..., x_n\}, v \in \{0,1\}\}_{\leq}$, where $\{\}_{\leq}$ is the sorted multiple-set in non-decreasing order. $f_{z=v}$ is the number of satisfying count of f when input z is set to constant v.

Figure 4b shows the count of 1-minterms (orange shadow), indicating the number of satisfying instances when input x_2 is set to 1. This signature captures the face characteristics within the Boolean function hypercube.

Definition 2 (Ordered sensitivity vector, OSV)**.** For a Boolean function f with all input patterns X in its truth table T(f), an ordered sensitivity vector is $OSV(f) = \{sen(f,X) : X \in \{0,1\}^n\}_{\leq}$. $sen(f,X) = |i : f(X) \neq f(X^i)|$ is the number of input literals that are sensitive for input pattern X, where $f(X^i)$ means negating the i-th variable in X.

Figure 4c shows the sensitivity for the input pattern 011. This signature captures the point characteristics of the Boolean function in hypercube. Trivial inputs that do not affect the function's output can be identified if negating an input does not alter the output across all input patterns.

Definition 3 (Ordered influence vector, OIV)**.** We denote the ordered influence vector as $OIV(f) = \{inf(f,z) : z \in \{x_1, x_2, ..., x_n\}\}_{\leq}$. $inf(f,i)$ is the influence of input x_i, which is defined as $\frac{1}{2^n}|f(X) \neq f(X^i) : X \in \{0,1\}^n|$.

The ordered influence vector models the sensitivity probability at x_i for input pattern X. It provides insights into the sensitivity relationship between opposing faces, as shown in Figure 4d. This signature captures the characteristics of the function hypercube's faces and points.

As encoding selection is a simple classification task, we employ a concise and lightweight model to select encoding for each Boolean function based on extracted features. We choose the MLP model with three 64-neuron internal layers. The first two layers use the Rectified Linear Unit (ReLU) activation function, and the third uses softmax. We set the learning rate to 0.02, the dropout rate to 0.2, the batch size to 32, and use Cross Entropy [15] as a loss function. We present two models in this paper: BESWAC-EF, which chooses a combination of encoding and symmetry-breaking methods, and BESWAC-EC, which focuses solely on identifying the suitable encoding method. Though lightweight, the MLP model achieves high accuracy, as validated by experiments in Section IV. The training process for this simple model converges quickly within a few hundred iterations. During inference, feature extraction takes negligible time compared to the SAT solver. By selecting the suitable encoding method, SAT solvers can operate more wisely and effectively than the original flow. This technique significantly accelerates the entire process while adding negligible execution time.

D. Early Exit

Section II reveals that UNSAT trials significantly contribute to the overall execution time. UNSAT trials cannot yield useful results, and it is intuitive to abandon such trials if possible. For exact synthesis, we identified empirical rules to detect challenging CNFs that cause long execution. We propose BESWAC-R, an optional early exit technique leveraging these empirical rules. When a CNF is identified as difficult, we immediately exit the SAT solver call and increment r for subsequent trials. BESWAC-R is an aggressive acceleration technique that may introduce suboptimality, though such occurrences were rare in our experiment. While BESWAC-R may deviate from the "exact" synthesis process by using more resources, the benefits of accelerated execution through early exiting are significant. Users can adjust this trade-off according to their requirements.

Our empirical rule considers two metrics: the ratio of CNF variables to clauses ($R_{CV} = \frac{\#clauses}{\#variables}$) and an execution time threshold (t_m). Previous research [16] suggests that CNFs on the boundary between SAT and UNSAT with $4.15 \leq R_{CV} \leq 4.55$ are challenging to solve. However, for exact synthesis problems, we find CNFs with R_{CV} ranging from 2.3^n-5 to 2.35^n+7 are hard to solve, where n is the input size. If a CNF falls within this range and the solving time exceeds the threshold t_m, we terminate the SAT solver to reduce

Encoding	4-input		5-input		6-input	
	R_{CV}	t_m	R_{CV}	t_m	R_{CV}	t_m
SSV	[24, 30]	200ms	[55, 70]	1.2×10^4s	[135, 150]	2.2×10^5s
MSV	[28, 43]	180ms	[85, 135]	1.1×10^4s	[230, 280]	1.6×10^5s
DITT	[30, 50]	240ms	[75, 145]	2.3×10^4s	[185, 215]	3.7×10^5s

Table I: Fine-grained empirical thresholds R_{CV} and t_m.

Relative Execution Time	4SSV NARCOS	4MSV NARC	4DITT NARCS	5SSV NARCOS	5MSV NARC	5DITT NARCS	6SSV ARCS	6MSV NARCS	6DITT NARCS	Average Speedup
Original Flow	1.00 ($478ms$)	1.10	1.11	1.08	1.00 ($1.67E4s$)	1.89	1.72	1.00 ($1.10E5s$)	3.54	1.00×
BESWAC-I	0.80	0.83	0.84	1.07	0.86	1.86	1.69	0.97	3.51	1.11×
BESWAC-E		0.87			0.48			0.53		1.59×
BESWAC	0.72 (**1.39× Speedup**)			0.43 (**2.33× Speedup**)			0.52 (**1.92× Speedup**)			**1.79×**
BESWAC-R (Optional)	0.97	1.06	1.05	0.58	0.55	1.04	1.24	0.73	2.51	1.34×

Table II: Relative execution time of BESWAC and other exact synthesis flows.

SSV (S)	NARCS	(1)	NARCOS	(2)	ARCS	(3)
	NOS	(4)	NARCO	(5)	ARS	(6)
MSV (M)	NARCS	(1)	NARCOS	(2)	NARC	(3)
	AOS	(4)	NARS	(5)	NACOS	(6)
DITT (D)	NARCS	(1)	NRCS	(2)	AO	(3)
	NCOS	(4)	NCO	(5)	NRS	(6)

Table III: Encoding symbols and symmetry-breaking method symbols.

Inputs	**Encoding Symbols**	**Symmetry-Breaking Symbols**						**Ave.**
		1	2	3	4	5	6	
4-input	S	20.3%	20.3%	20.2%	21.7%	22.4%	21.9%	22.9%
	M	24.8%	24.7%	24.2%	23.9%	24.2%	24.4%	
	D	24.8%	24.7%	20.2%	24.3%	24.0%	20.3%	
5-input	S	8.3%	12.7%	9.5%	2.8%	1.8%	1.6%	5.8%
	M	11.9%	6.3%	14.3%	4.5%	3.1%	7.9%	
	D	1.6%	5.4%	0.8%	5.3%	7.4%	0.2%	
6-input	S	2.2%	1.9%	1.8%	0.3%	0.1%	0.6%	1.4%
	M	3.1%	2.5%	7.4%	0.8%	0.7%	0.1%	
	D	0.6%	1.1%	0.9%	0.1%	0.3%	0.4%	

Table IV: Execution time reduction by initial value prediction.

overall execution time. The chosen threshold values are $t_m = 250ms, 2.6 \times 10^4 s, 3.4 \times 10^5 s$ for 4-input, 5-input, and 6-input functions respectively. The threshold may vary depending on the SAT solver and machine performance. Skipped trials that meet the early exit criteria require significantly more time than other cases. In addition to generally applied rules (BESWAC-RC), we also design fine-grained rules (BESWAC-RF) for different encoding methods of Boolean functions, as detailed in Table I. Experimental results in Section IV demonstrate the effectiveness of early exit with tolerable suboptimality.

IV. Evaluations

BESWAC is implemented in C++ and Python with ParKissat [8] as the SAT solver. We compare BESWAC with state-of-the-art exact synthesis flows using encoding and symmetry-breaking methods, as summarized by Haaswijk et al. [4]. The benchmark consists of 30,000 randomly selected Boolean functions each for 4-input, 5-input, and 6-input categories. For each function, we limit SAT solving for a single case to 2E07s and rule out cases exceeding the time limit. All the experiments are evaluated on a server with 2-way Intel Xeon Gold 6248R CPUs and 512GB DDR4 memory. It is worth mentioning that we remove topology information for a fair comparison. However, as discussed in Section I, BESWAC is orthogonal to existing methods. Therefore, if desired, previous methods like topology information can be combined with BESWAC for further acceleration.

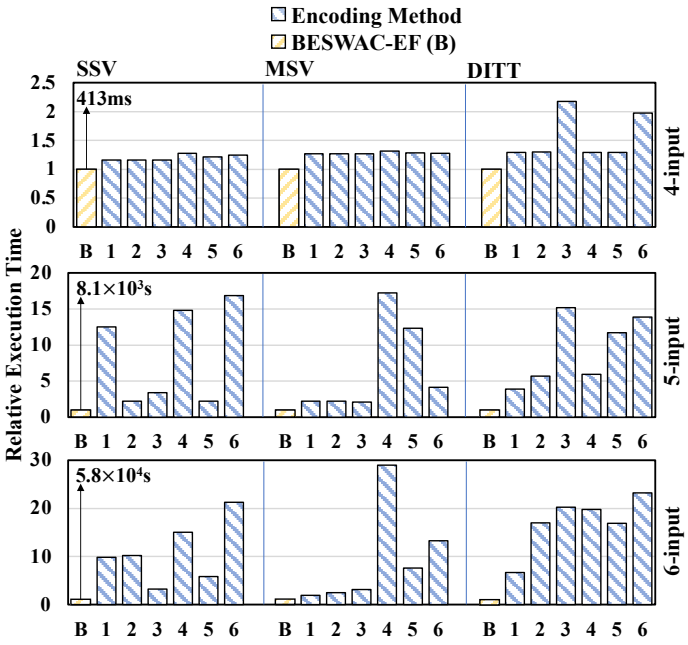
Accuracy	**BESWAC-EC**	**BESWAC-EF**
4-input Functions	99.8%	75.7%
5-input Functions	98.4%	78.0%
6-input Functions	96.9%	71.2%

Table V: Accuracy of encoding selection model in BESWAC.

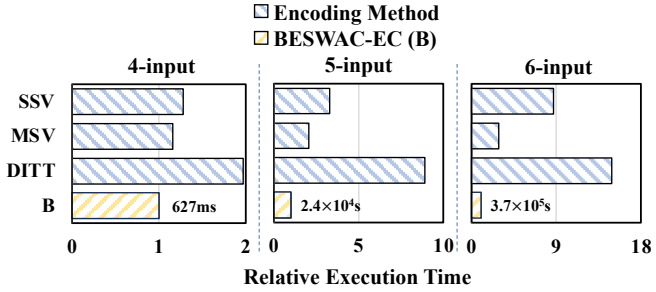
Table II demonstrates the effectiveness of BESWAC in boosting the exact synthesis flow. It compares BESWAC and the original exact synthesis flow using encoding and symmetry-breaking methods with the best performance [4]. BESWAC significantly reduces the execution time of the exact synthesis flow. On average, it achieves an average speedup of 1.39×, 2.33×, and 1.92× for 4-input, 5-input, and 6-input functions, respectively. And the overall speedup reaches 1.79×. This speedup benefits the on-the-fly rewriting and expedites the construction of optimal circuit libraries. The three modules in BESWAC: initial value prediction (BESWAC-I), encoding selection (BESWAC-E), and optional early exit (BESWAC-R) contribute to execution time reduction independently, where only the BESWAC-R may cause rare and tolerable quality loss. They achieve 1.11×, 1.59×, and 1.34× average speedups, respectively. In the following evaluations, we further analyze the effect of each module in detail.

For simplicity, we use the symbols listed in Table III to represent the combination of encoding and symmetry-breaking methods. For example, the combination of SSV encoding and NARCS symmetry-breaking methods is denoted as S-1. We apply lower bounds in Section III-B to 4-input, 5-input, and 6-input Boolean functions to evaluate initial value prediction. Results in Table IV demonstrate that BESWAC-I effectively reduces execution time across all combinations of encoding and symmetry-breaking methods. On average, BESWAC-I saves 22.9%, 5.8%, and 1.4% time for 4-input, 5-input, and 6-input functions, respectively. The best results reach up to 24.8%, 14.3%, and 7.4%, highlighting the effectiveness of initial value prediction in reducing execution time.

For encoding selection, we select the combinations of encoding and symmetry-breaking methods listed in Table III as candidates. The ranks of sampled execution time using these encoding methods work as the ground truth for the encoding selection model. The Boolean functions are divided into training and testing sets randomly and exclusively. The training set contains 4,000 functions for each input size, a small proportion of all Boolean functions. The training process from scratch converges within 4 hours. Table V shows that BESWAC-EF and BESWAC-EC accurately predict suitable encoding methods for Boolean functions, with an average accuracy of over 71% and



(a) Fine-grained selection.



(b) Coarse-grained selection.

Figure 5: Boosting effect of encoding selection in BESWAC.

Inputs	BESWAC-RC			BESWAC-RF		
	Time Saved	r_1	r_2	Time Saved	r_1	r_2
4-input	2.5%	4.5%	2.0%	4.1%	3.8%	2.7%
5-input	21.7%	3.6%	2.8%	46.2%	2.3%	1.7%
6-input	37.5%	3.9%	1.3%	26.9%	3.3%	1.4%

Table VI: Results of early exit technique in BESWAC.

96%, respectively. Accurate selection helps reduce execution time in the exact synthesis flow, as shown in Figure 5. On average, BESWAC-EF achieves a speedup of 1.16×, 2.07×, and 1.89× for 4-input, 5-input, and 6-input functions, respectively, compared with best-performance combinations. The coarse-grained version BESWAC-EC (no symmetry-breaking clauses) achieves a speedup of 1.16×, 2.06×, and 2.90×. The results affirm BESWAC's informative feature design and effective encoding selection, which enables wiser SAT solver calls.

We sampled 500 functions for optional early exit to build the R_{CV} and t_m parameters. We test coarse-grained empirical rules (BESWAC-RC) and fine-grained rules (BESWAC-RF) with results in Table VI. On average, BESWAC-RC saves 2.5%, 21.7%, and 37.5% time for 4-input, 5-input, and 6-input functions, respectively, while BESWAC-RF performs even better with time savings of 4.1%, 46.2%, and 26.9%. For this optional technique, the maximum quality loss was two gates, with $r_1 = 4.6\%$ and $r_2 = 2.8\%$ for BESWAC-RC, where r_i means an i-unit loss compared to optimal results. Fewer circuits experience losses when using fine-grained rules, with $r_1 = 3.8\%$ and $r_2 = 2.7\%$. Users can consider this technique, weighing time savings against potential but rare quality loss.

V. Conclusion

In this paper, we propose BESWAC to boost the exact synthesis framework at the flow level by reducing the execution time of the trial-and-error flow. It consists of three key modules: initial value prediction to reduce SAT solver calls, encoding selection to offer instance-specific solver-friendly encoding, and optional early exit to abandon hard trials timely. These modules improve the efficiency of the exact synthesis process, benefiting faster optimal library generation and on-the-fly rewrite for logic synthesis. Evaluations demonstrate that BESWAC achieves an average speedup of 1.79× compared to the original flow. Besides, BESWAC is orthogonal to the existing boosting methods, making it a practical addition to the exact synthesis flow.

Acknowledgement

This work was partly supported by the National Natural Science Foundation of China (Grant No. 62090021) and the National Key R&D Program of China (Grant No. 2022YFB4500500).

References

[1] H. Riener *et al.*, "On-the-fly and DAG-aware: Rewriting Boolean networks with exact synthesis," in *DATE*, 2019.
[2] W. Haaswijk *et al.*, "Classifying functions with exact synthesis," in *International Symposium on Multiple-Valued Logic (ISMVL)*, 2017.
[3] X. Wang *et al.*, "MinSC: An exact synthesis-based method for minimal-area stochastic circuits under relaxed error bound," in *ICCAD*, 2021.
[4] W. Haaswijk *et al.*, "SAT-based exact synthesis: Encodings, topology families, and parallelism," *IEEE TCAD*, 2019.
[5] A. Kojevnikov *et al.*, "Finding efficient circuits using SAT-solvers," in *The International Conference on Theory and Applications of Satisfiability Testing (SAT)*. Springer, 2009, pp. 32–44.
[6] D. E. Knuth, *The Art of Computer Programming, Volume 4B: Combinatorial Algorithms*. Addison-Wesley Professional, 2022.
[7] X. Ge *et al.*, "Topology-based exact synthesis for majority inverter graph," in *ISCAS*, 2022.
[8] X. Zhang *et al.*, "ParKissat: Random shuffle based and pre-processing extended parallel solvers with clause sharing," *SAT Competition*, 2022.
[9] Z. Chu *et al.*, "Advanced functional decomposition using majority and its applications," *IEEE TCAD*, 2019.
[10] E. Demenkov *et al.*, "New lower bounds on circuit size of multi-output functions," *Theory of Computing Systems*, 2015.
[11] J. Li *et al.*, "3.1n - o(n) circuit lower bounds for explicit functions," in *STOC*, 2022.
[12] A. Golovnev *et al.*, "Circuit size lower bounds and sat upper bounds through a general framework," in *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2016.
[13] A. Mishchenko *et al.*, "DAG-aware AIG rewriting: A fresh look at combinational logic synthesis," in *DAC*, 2006.
[14] J. Zhang *et al.*, "Rethinking npn classification from face and point characteristics of boolean functions," in *DATE*, 2023.
[15] Z. Zhang *et al.*, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *NeurIPS*, 2018.
[16] D. Mitchell *et al.*, "Hard and easy distributions of SAT problems," in *AAAI*, 1992.