2025 Design, Automation & Test in Europe Conference (DATE 2025)

Hybrid Exact and Heuristic Efficient Transistor Network Optimization for Multi-Output Logic

Lang Feng[*] Rongjian Liang[†] Hongxin Kong[‡]
[*]Sun Yat-sen University, China [†]NVIDIA, USA [‡]Texas A&M University, USA
fenglang3@mail.sysu.edu.cn; rliang@nvidia.com; konghongxin@outlook.com

***Abstract*—With the approaching post-Moore era, it is becoming increasingly impractical to decrease the transistor size in digital VLSI for better performance. To address this issue, one approach is to optimize the digital circuit at the transistor level to reduce the transistor count. Although previous works have explored ways to conduct transistor network optimization, most of these efforts have focused on single-output networks or applied heuristics only, limiting their scope or optimization quality. In this paper, we propose an exact transistor network optimization algorithm that supports multi-output logic and is formulated as a SAT problem. Our approach maintains a high optimization level by employing the exact algorithm, while also incorporating a hybrid process that uses a heuristic algorithm to predict the solution range as a guidance for better efficiency. Experimental results show that the proposed algorithm has a 5.32% better optimization level given 54% less runtime compared with the state-of-the-art work.**

***Index Terms*—Transistor Networks, SAT, Heuristic Algorithm**

I. Introduction

The shrinking of transistor sizes has greatly improved VLSI performance over the years. However, physical limitations have been encountered with technology below 7nm. To address this, new approaches are being explored to further enhance VLSI performance. One potential direction is the optimization and expansion of the standard cell library, which offers more options for logic synthesis. This expansion can broaden the design space during synthesis and improve performance. Given the variety of standard cells, algorithms [1]–[8] for transistor network optimization have been proposed to reduce the transistor count and efficiently provide the standard cell structures.

In modern designs, CMOS technology is the most popular for digital standard cell implementation, where each standard cell contains PMOS and NMOS networks. However, the complexity of CMOS transistor network optimization is significantly large as the transistor network does not follow the regular rules of digital circuit design, especially when considering non-serial parallel (NSP) network structures. Given a standard cell's logic function, previous work proposes different ways to automatically construct the networks, including logic function optimization [1]–[4], [9]–[11], graph-based optimization [5]–[8], [12], [13], SAT-based optimization [14], etc. However, previous work typically only supports single-output logic or relies only on heuristics for optimization, leaving room for further improvement in network structure scope and quality.

[*]The corresponding author. [‡]The idea was in part from Dr. Kong when he was in Texas A&M University. [+]This work was partially supported by National Natural Science Foundation of China (Grant No. 92473207 and 62204111).

In order to address the challenge above, this work introduces a transistor network optimization algorithm for CMOS standard cells that can handle multi-output logic functions. This algorithm is SAT-based, offering a theoretically superior optimization level compared to heuristic algorithms. Additionally, the work proposes a hybrid approach that combines the heuristic algorithm for reference to dynamically guide the SAT-based algorithm for faster regression. By integrating these approaches, the proposed algorithm is able to produce smaller networks with shorter runtimes compared to previous state-of-the-art work. The key contributions of this work are as follows:

- We proposed a novel CMOS transistor network optimization algorithm that can handle multi-output logic under the exact modeling, which, to the best of our knowledge, is the first work of its kind. By formulating the problem as a SAT problem and utilizing the SAT solver, our work has achieved a high level of theoretical optimization with a lightweight transistor count.
- We introduced binary search as the searching order of the transistor sizes in the SAT formulations, and proposed a hybrid process to adjust the binary search boundaries at runtime. This approach utilizes other heuristic network optimization algorithms as references, helping the proposed algorithm to quickly converge to the solution while maintaining a high level of optimization.
- The experiments demonstrate that the proposed algorithm can generate smaller transistor networks within less runtime than the previous state-of-the-art approach for multi-output logic functions. Additionally, it shows that the hybrid process significantly improves the search speed.

II. Previous Works

Transistor network optimization has been a subject of research for a long time. Recently, with the arrival of the post-Moore era, it has garnered increased attention. The transistor network can be typically generated given a logic function with only AND, OR logic (with literals with or without NOT logic), which indicates serial and parallel connections (i.e., serial-parallel (SP) network, with an example in Figure 1(a)). In this case, optimizing the logic functions can directly reduce the transistor counts [1]–[4], [9]–[11]. For example, the work [1] optimizes the sequential circuits' transistor counts by optimizing the logic function, but the optimization is only performed on gate-level. The works [2]–[4] all propose algorithms to simplify the boolean functions, especially that the work [4] is able to perform simplification given multiple objectives, which can be leveraged for transistor network generation.

However, the transistors can be connected with transistor bridges, leading to the non-serial-parallel (NSP) networks (An example is in Figure 1(b)). This increases the search space and optimization difficulties. Graph-based optimization [5]–[8], [12], [13] and SAT-based optimization [14] are proposed to tackle this challenge. For example, the work [7] uses the graph pattern heuristics to simplify the transistor network graph. The work MOTO-X [8] gradually builds the transistor bridges given each product term in the logic function and checks the bridge's legibility. Different from most of the previous works, MOTO-X is able to handle multi-output logic functions and outperforms previous works [15], [16], but its heuristic nature still restricts the solution quality. This is the key difference to the proposed approach of this paper, of which the formulation is exact. A recent work MiniTNtk [14] proposes a SAT-based transistor network optimization. By leveraging the SAT solvers, the results are highly optimized, but MiniTNtk cannot handle multi-output functions, and it does not investigate enough improvement on the efficiency. In contrast, this work proposes a hybrid exact and heuristic transistor network optimization algorithm. It not only can handle multi-output functions with high optimization level, but also is more efficient compared with the state-of-the-art work MOTO-X supporting multi-output functions.

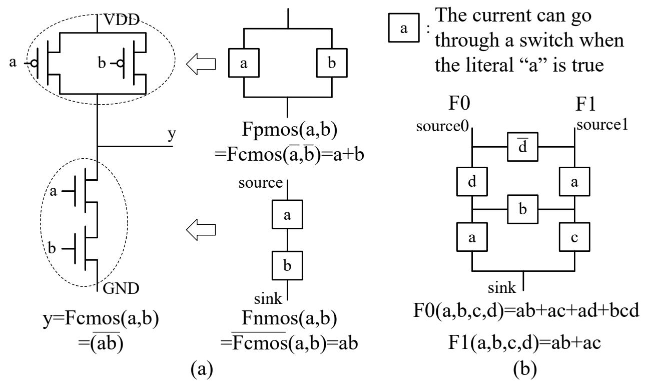
III. Preliminaries

A. CMOS Transistor Network Optimization

A CMOS circuit comprises pull-up and pull-down networks, which are constructed by PMOS and NMOS transistors, respectively. Typically, the transistor network optimization algorithms generate PMOS and NMOS networks separately [7], [8], and then combine them for the CMOS circuits. The objective is usually to minimize the transistor count with the critical path transistor count constraints [7], [8], [14]. When considering the NMOS part of a CMOS circuit (The PMOS part is similar), it can be processed as a switch network G_{swtich} with multiple source nodes and one sink node. Each node stands for a switch (implemented by a transistor), and each edge stands for an electrical connection of switches. G_{swtich} corresponds to a logic function F if and only if a pair of source and sink is electrically connected/unconnected when an input pattern of F is true/false, respectively. Note that each source node corresponds to one output function. Each switch in a switch network is associated with a literal (x or $\overline{x}$), and it is electrically connected when the literal is true. Examples of the switch networks for a complete CMOS circuit and an NMOS network (which is a NSP network in this example) for multi-output logic are shown in Figure 1(a) and (b).

B. SAT-based Single-Output Transistor Network Optimization

We applied the MiniTNtk [14] framework for further exploration, and its SAT formulation is first briefly introduced here. Its SAT formulation is to output a switch network (represented by SAT variables) given the number of the switches (transistor count) r and a logic function. Based on the SAT formulation,



VDD
a
b
y
GND
y=Fcmos(a,b) =(ab)
a
b
Fpmos(a,b) =Fcmos(a,b)=a+b
source
a
b
sink
Fnmos(a,b) =Fcmos(a,b)=ab
(a)
a : The current can go through a switch when the literal "a" is true
F0
F1
source0
source1
d
d
a
b
a
c
sink
F0(a,b,c,d)=ab+ac+ad+bcd
F1(a,b,c,d)=ab+ac
(b)

Fig. 1. (a) An example of the switch networks from a two-input NAND gate; (b) An example of a NMOS switch network with multiple outputs.

MiniTNtk flow is to linearly increase r by 1 (named linear search in the following) until r switches can satisfy the given logic function with the optimal transistor count.

The SAT formulation of MiniTNtk can be explained by Figure 2. The first step is to form the main decision variables. Assuming that the given function is $F(a, b) = \overline{a}b$, and the given transistor count r to be tried is 2, a switch graph (G_{switch}) can be built as Figure 2(a). A source node 0 and a sink node 5 are introduced as terminals. Each switch shown as a rectangle introduces 2 nodes i and j (the edge between i and j is named a *switch edge*), and is controlled by literal l_{ij}. Each edge between nodes i and j without crossing switches is named an *internal edge*, of which the connecting status is x_{ij}. The graph is initialized with all possible candidate internal edges. Then, l_{ij} and x_{ij} are the decision variables, with $x_{ij} = 1$ indicating the selection of the corresponding candidate edge, and $l_{ij} = e$ indicating the selection of literal e to control the switch. The decision variable l_{ij} discussed can be formed into multiple binary variables in the SAT problem [14]. In Figure 2(a), one solution is shown, with literals marked inside the switches, and solid lines representing the selected candidate edges.

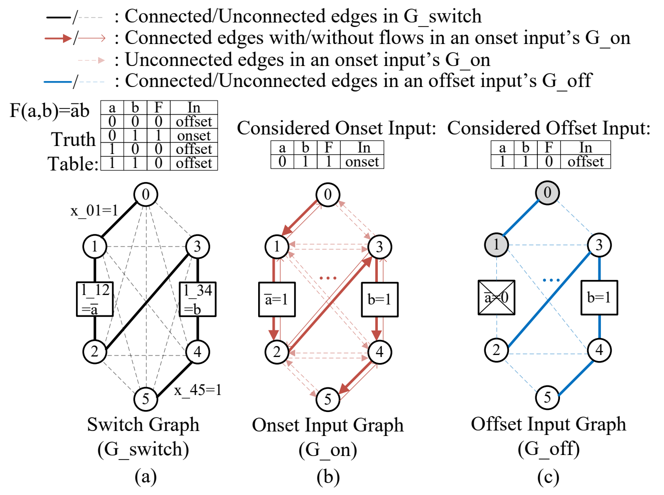


Fig. 2. An example for single-output SAT formulation of MiniTNtk. (Some edges are hided as "...".)

The second step is to form the constraints, which are formed according to the truth table. The input patterns with output

1 form the "onset", otherwise they form the "offset". The constraints of MiniTNtk rely on the fact that if and only if an input belongs to the onset, there exists a path from the source node to the sink node. For an onset input, the constraints can be built by constructing an onset input directed graph G_{on}, where the nodes are the same as G_{switch}, and each edge in G_{switch} is replaced by 2 direct edges between the same pair of nodes in G_{on}. This is to represent all possible paths. G_{on} is then built as a flow network, and the flow can pass edges between nodes if and only if the edges between the same nodes are connected in G_{switch}. In this case, if and only if there is a flow from a source to the sink in G_{on}, it means that they are electrically connected under the corresponding input pattern. This can be achieved by SAT constraints [14]. Combining all constraints for the onset input patterns, the SAT formulation ensures connectivity when the logic function's output is 1. Additionally, the constraints guarantee that the critical path is $\leq$ M transistors. In contrast, for each offset input pattern, there must not exist a flow from the source to the sink. This forms as a node partitioning task, and an offset input undirected graph G_{off} is built for each offset input, shown in Figure 2(c). Constraints can be built to ensure that nodes electrically connected in G_{switch} are at the same part, while the source and sink nodes are fixed in different parts. An example of partitioning is shown as different colors of nodes in Figure 2(c). Similar to onset inputs, each offset input builds a separated G_{off}. The detailed equations of MiniTNtk's SAT formulation can be further found in the work [14].

IV. Overview of the Proposed Hybrid Algorithm

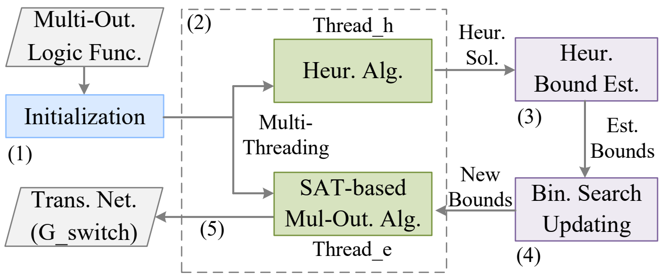


Fig. 3. The overview of the proposed hybrid algorithm.

We first introduce the overall flow of the proposed hybrid algorithm in Figure 3. This algorithm takes a multi-output logic function as input and produces a transistor network G_{switch} as output. Each step in the flow can be briefly described as follows.

(1) *Initialization:* Given a multi-output logic function (as well as the critical path transistor count constraint), the proposed algorithm estimates the lower and upper bounds of the binary search, which is prepared for the SAT-based multi-output transistor network optimization algorithm.
(2) *Parallel Hybrid Searching:* After step (1) finishes, two parallel threads are created, one for the exact algorithm (which is the SAT-based multi-output transistor network optimization) and another for the heuristic algorithm.
(3) *Heuristic Bound Calculation:* Once there is a solution from the heuristic algorithm in step (2), a pair of new bounds for the binary search is estimated according to this solution.
(4) *Binary Search Updating:* Based on the estimated bounds in step (3) and the current status of the SAT-based optimization in step (2), the binary search bounds are updated, and the SAT-based process is adjusted if needed.
(5) *Solution Output:* Once a better G_{switch} is found, it is outputted, until the exact algorithm finishes.

In Section V, the SAT-based multi-output logic transistor network optimization in step (2) is proposed. In section VI, the complete flow is elaborated.

V. Exact Transistor Network Optimization of Multi-Output Logic

In this section, the SAT-based multi-output transistor network optimization shown in step (2) of Figure 3 is proposed. Multi-output logic is commonly found in digital VLSI designs, such as full adders. Optimizing the transistor network for multi-output logic can involve sharing transistors among different outputs to reduce the overall transistor count. However, it can be challenging to design a network that avoids conflicts between different logic functions while allowing the transistors required by these functions to be shared. One approach is to create two separate networks to implement the two functions in order to avoid conflicts, but this results in a relatively larger transistor count. Alternatively, the proposed approach may maintain the main structure of both networks while generating each output network, differing only in the output source nodes. However, after merging the main structures, new invalid paths may exist across the output nodes' edges.

In order to address this challenge, we propose an algorithm for optimizing a transistor network that can handle multiple outputs. This algorithm models all terminal connections for the outputs in a single network, taking into account the input patterns for all output functions. An example of this is illustrated in Figure 4, which shows two logic functions F_0 and F_1 for two outputs. The problem can be formulated as a SAT problem, with the data structure (the graph) inspired by MiniTNtk. It is important to note that the source nodes for outputs 0 and 1 (node 0 and node 0') are part of the same switch graph. The solution is shown in Figure 4(a). When building constraints based on G_{on}/G_{off} of each onset/offset input, the source node of the constraints is chosen according to the output of the onset/offset input, respectively. In this scenario, the unchosen source nodes are considered regular nodes.

In Figure 4(b), two G_{on} of two onset inputs are displayed under different outputs (F_0 and F_1). When considering an onset input of F_0, the source node is node 0, and SAT constraints are built to ensure that there is an electrical connection between node 0 and the sink (node 9), while node 0' is a regular node. On the other hand, when considering an onset input of F_1, node 0' is treated as the source node, and node 0 is a regular node. When the SAT constraints are built with each G_{on} similar to the single-input logic SAT formulation, the connections between source and sink pairs through different output source nodes can be simultaneously considered to avoid conflicts. This is because each G_{on}/G_{off} contains all output nodes in one connected graph, instead of separating output nodes in multiple sets of graphs as Figure 4(b).

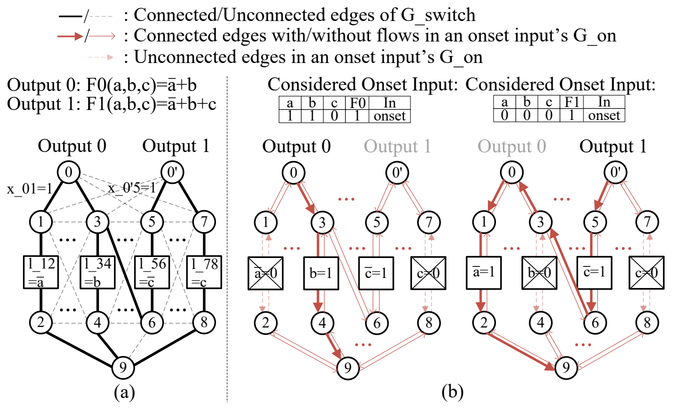

: Connected/Unconnected edges of G_switch
: Connected edges with/without flows in an onset input's G_on
: Unconnected edges in an onset input's G_on
Output 0: F0(a,b,c)=ā+b
Output 1: F1(a,b,c)=ā+b+c
Considered Onset Input:
Considered Onset Input:
Output 0
Output 1
x_01=1
x_0'5=1
(a)
(b)

Fig. 4. (a) An example of G_{switch} for building SAT formulation of a 2-output logic; (b) The corresponding G_{on}s of 2 onset inputs for a 2-output logic.

When building each G_{off}'s SAT constraints, the partitioning problem is built so that the source node corresponding to the considered output should be at a different part from the sink node. The source node for the output not being considered is treated as a regular node that could be in either part.

Given the above-proposed algorithm, the basic flow for SAT-based multi-output transistor network optimization is to do a linear search for the transistor count from a lower bound until a satisfied network is found for an optimal transistor count.

VI. Hybrid Exact and Heuristic Process

If the proposed exact multi-output transistor network optimization conducts linear search ($O(optimal\_count - lower\_bound)$ iterations), it can lead to large runtime. In this work, we apply the binary search ($O(log(upper\_bound - lower\_bound))$) for higher speed. More importantly, a hybrid process that allows another heuristic algorithm to get involved to help the efficiency of the regression. This is based on the following assumptions: 1) Although heuristic algorithms for optimizing transistor networks may limit the level of optimization, they are typically fast enough to provide a coarse-grained solution. 2) By running more iterations, the heuristic algorithms will converge to a transistor count that is no better than the exact algorithm. 3) Meanwhile, the optimal transistor count is not far from the count of a well-designed heuristic algorithm, given enough time. Based on assumptions 1) and 2), the transistor counts through time from a heuristic algorithm can be illustrated as the curve marked by "Heuristic Solution" in Figure 5. Besides, based on assumption 3), a transistor count curve that gives a lower bound can be estimated and closer to the heuristic solution curve with a larger runtime. Note that the lower bound estimated here might not be precise, for which we compensate by conducting additional search if necessary (detailed later in this section).

By leveraging the heuristic solution and the estimated lower bound from it, the binary search of the transistor count can be sped up. An example is Figure 5. The detailed hybrid algorithm can be described as follows.

(1) Initialization: Given the output function set, the critical path's transistor counts M, and a heuristic algorithm that satisfies assumptions 1)-3), the hybrid algorithm can start. The

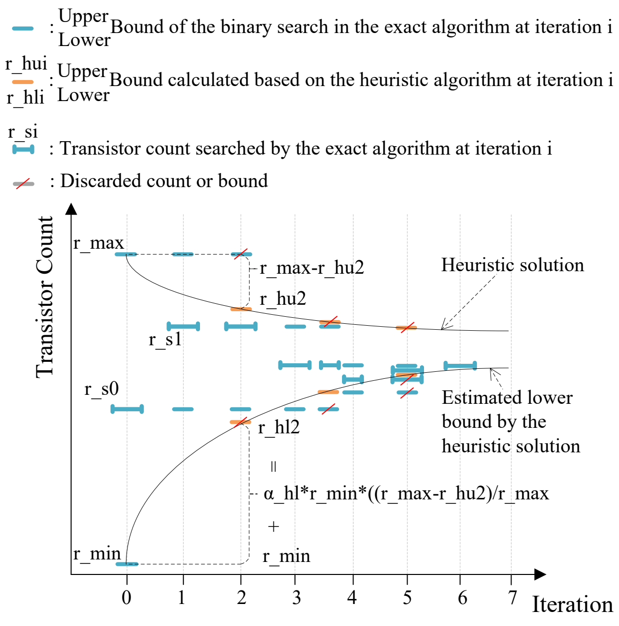


Fig. 5. An example of searching the optimized transistor count by hybridizing with the heuristic algorithm.

first step is to quickly run the heuristic algorithm to obtain a transistor count, which is used as the global upper bound r_{max}. Meanwhile, a global lower bound is also generated by $r_{min} = \alpha_{min} \times r_{max}$, where α_{min} is a parameter < 1 set by the users (it is set as 0.5 in our experiments). Following this, two threads $Thread_e$ and $Thread_h$ are created for the proposed exact multi-output logic algorithm, and the inputted heuristic algorithm, respectively. The initial upper and lower bounds for the exact algorithm's binary search are set as r_{max} and r_{min}, respectively, and the middle point r_{s0} is searched at first.

(2) Parallel Hybrid Searching: After $Thread_e$ and $Thread_h$ are created, the returns of either thread are periodically checked.

(3) Heuristic Bound Calculation: When the heuristic algorithm at $Thread_h$ returns a new solution during its iteration, the upper r_{hu} and lower r_{hl} bounds estimated from the heuristic solution are calculated as follows. When determining the value of r_{hu}, it is directly set as the transistor count in the solution from $Thread_h$, as the optimal solution is no worse than the current heuristic solution. As for r_{hl}, when r_{hu} gradually decreases with more iterations of the heuristic algorithm, r_{hu} should become tighter and the optimal solution should get closer to r_{hu} than before. Consequently, the lower bound r_{hl} should also gradually increase to approach r_{hu}. A heuristic concept is that r_{hl}'s distance to r_{min} should be proportional to the distances between r_{hu} and r_{max}. In this case, it is modeled as $r_{min} \times \frac{r_{max} - r_{hu}}{r_{max}}$, and a parameter α_{hl} can be used to control the scale. After this, r_{hu} and r_{hl} are checked to make sure $r_{hu} \geq r_{hl}$. An example is r_{hu2} and r_{hl2} in Figure 5.

(4.1) Binary Search Bound Updating: If either r_{hu} or r_{hl} is tighter than the current binary search bounds, the binary search bounds are adjusted to r_{hu} and r_{hl}, respectively. For instance, the discarded upper bound at iteration 3 is adjusted to r_{hu2}. Similar adjustments can be found between iterations 3 and 4 of the SAT-based algorithm. The bound updating can

accelerate the search process of the SAT-based algorithm in the later iterations.

(4.2) Search Point Updating: If the current transistor count r_s for the proposed SAT-based algorithm thread $Thread_e$ exceeds the new updated bounds, $Thread_e$ is canceled. Then, the middle point r_s of the binary search is recalculated, and a thread $Thread_e$ with updated inputs is created. An example is r_{s5} in Figure 5.

(5) Solution Output: The proposed SAT-based algorithm with binary search and the heuristic algorithm are conducted simultaneously. The best G_{switch} is outputted when improved solutions are found. It's important to note that for optimality, if the final iteration of the SAT-based algorithm reaches the lower bound of the binary search and it is still satisfiable, the lower bound of the binary search is decreased by 1 and the search continues until it is no longer satisfiable.

The hybrid transistor network optimization algorithm combines the advantages of both exact and efficient heuristic algorithms. An example is shown in Figure 6, where the search traces are illustrated. Linear search requires a large number of iterations to complete, while binary search halves the size of the searching range and finally regresses at iteration 4. In contrast, guided by the curve of the heuristic algorithm, the hybrid process can have a better estimation of the middle point at iteration 1. This can result in earlier regression at iteration 3, which is the most efficient.

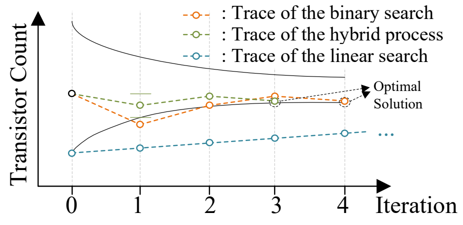


Fig. 6. Example traces of 3 searching orders (Black curves are the estimated bounds by heuristic solutions as Figure 5).

VII. Experimental Results

A. Experiment Setup

The experiments are conducted on a server with Intel Xeon Gold 6348 CPU and 512 GB memory. We use Gurobi 11.0.1 [17] as the SAT solver. Note that Gurobi is primarily a mixed integer programming (MIP) solver, but SAT problem can always be mapped to an MIP problem, with each SAT clause in conjunctive normal form (CNF) $(x_0 \vee x_1 \vee x_2 ...)$ mapped to an MIP constraint $x_0 + x_1 + x_2 + ... \geq 1$, and each x_i is a binary variable. Additionally, each $\overline{x_i}$ can be transformed to $(1 - x_i)$ in the MIP equation. Gurobi is a leading commercial solver for non-linear optimization, and can outperform many dedicated SAT solvers. We reimplement the state-of-the-art multi-output transistor network optimization algorithm MOTO-X [8] for comparison due to the unavailability of open-source implementation and previous total runtime reports. Note that MOTO-X is also a heuristic algorithm, it is used as the heuristic algorithm in the proposed hybrid optimization. The benchmark contains the testcases in the paper of MOTO-X [8], and the critical path transistor count M of each case is set so that the reimplemented MOTO-X's solution is similar to the paper [8]. Besides, although MiniTNtk [14] is not for multi-output cases, its linear searching order (which increases the transistor count by 1 for each iteration) is adopted for the comparison of the searching efficiency. Finally, we set a timeout feature for the SAT solver to restrict the upper bound of the runtime (5k-10k seconds) for each iteration.

B. Transistor Count Results and Comparisons

The main experimental results of the proposed algorithm and the comparison are shown in Table I, where the transistor count and runtime are tested, and the selection of these metrics follows the previous works [7], [8], [14]. In this table, "Binary" and "Hybrid" stand for the proposed multi-output logic transistor network optimization algorithm with using binary searching order and a hybrid process, respectively. Similar to "Hybrid", the initial upper and lower bounds of "Binary" are still estimated by the same heuristic algorithm, but "Binary" barely applies binary search without further guidance from the heuristic algorithm during searching. According to Section III, the CMOS circuits' networks are executed separately by PMOS and NMOS parts. The runtime is normalized to the total runtime of "Hybrid". Besides, the total transistor count of the CMOS is also shown in Table I, and the input and output numbers are shown near each testcase's name.

MOTO-X is compared in this experiment. Note that the number of MOTO-X's iterations can be controlled by users. It is set large enough to allow longer runtime than "Hybrid" to compare the transistor count optimality, and set much smaller to quickly provide the estimated bounds when MOTO-X is used as the heuristic algorithm in "Hybrid". Even MOTO-X is allowed to run for a longer duration, the final transistor count of each testcase is no better than the proposed algorithm "Hybrid". With a shorter runtime, the total transistor count of "Hybrid" is 5.32% better than the state-of-the-art multi-output logic optimization algorithm MOTO-X. This indicates the effectiveness and the better optimization limit of the proposed algorithm. Besides, when comparing to our proposed algorithm "Binary" without hybrid process, "Hybrid" has both better runtime and transistor count due to the hybrid effect and the guidance from the heuristic algorithm. It continuously updates the searching bounds during the regression of the heuristic algorithm. Even if the early stage estimation might provide inferior bounds, they are likely to be trimmed in the later stage. The higher average transistor count of "Binary" is due to the fact that the total runtime is set to be similar as "Hybrid", and "Binary" searches less efficiently. Note that for "AGR1" and "AGR2", the runtime of "Binary" is better with the same solutions as "Hybrid". This is due to the uncertainty in the heuristic algorithm to estimate the initial search bounds for both "Binary" and "Hybrid". In most cases and on average value, it is still shows the effectiveness of the hybrid process.

C. Runtime Analysis

We analyze the runtime from different perspectives. First is about the runtime comparison between the setup with and

TABLE I

THE TRANSISTOR COUNT AND RUNTIME UNDER MULTI-OUTPUT CASES [8] AND COMPARISONS WITH THE STATE-OF-THE-ART WORK [8].

Testcases (#in/#out)	MOTO-X [8]						Binary						**Hybrid**					
	Transistor Count (↓)			Normalized Runtime (↓)			Transistor Count (↓)			Normalized Runtime (↓)			Transistor Count (↓)			Normalized Runtime (↓)		
	PMOS	NMOS	Total	PMOS	NMOS	Total	PMOS	NMOS	Total	PMOS	NMOS	Total	PMOS	NMOS	Total	PMOS	NMOS	Total
BAR1(3/3)	11	11	22(28)	0.59	1.1	1.68	10	11	21(27)	0.53	0.95	1.48	10	11	21(27)	0.52	0.48	1.00
BAR2(4/3)	19	18	37(45)	0.87	0.6	1.47	13	13	26(34)	0.4	0.66	1.06	13	15	28(36)	0.44	0.56	1.00
AGR1(4/3)	14	14	28(36)	0.68	2.87	3.55	11	11	22(30)	0.36	0.49	0.85	11	11	22(30)	0.55	0.45	1.00
AGR2(4/2)	13	12	25(33)	0.49	1.12	1.61	11	9	20(28)	0.67	0.32	0.99	11	9	20(28)	0.38	0.62	1.00
AGR3(4/2)	21	14	35(43)	1.13	0.28	1.41	24	14	38(46)	0.61	0.41	1.02	20	14	34(42)	0.73	0.27	1.00
GUR1(4/3)	18	19	37(45)	0.81	1.63	2.44	27	22	49(57)	0.8	0.67	1.47	20	19	39(47)	0.6	0.4	1.00
GUR2(4/2)	8	9	17(25)	0.65	0.68	1.33	6	7	13(21)	0.49	0.5	0.99	6	7	13(21)	0.49	0.51	1.00
SPEC(4/4)	25	22	47(55)	0.92	1.5	2.42	32	31	63(71)	0.43	0.59	1.03	26	21	47(55)	0.5	0.5	1.00
STE1(3/2)	10	9	19(25)	2.8	0.67	3.47	9	9	18(24)	0.95	0.98	1.92	9	9	18(24)	0.5	0.5	1.00
STE2(4/2)	14	14	28(36)	0.58	0.65	1.23	22	12	34(42)	0.45	0.29	0.73	13	12	25(33)	0.51	0.49	1.00
STE3(3/3)	10	10	20(26)	0.81	1.3	2.1	9	8	17(23)	0.91	0.53	1.43	9	8	17(23)	0.45	0.55	1.00
STE4(4/5)	49	59	108(116)	1.14	0.95	2.09	67	67	134(142)	0.5	0.5	1	53	53	106(114)	0.5	0.5	1.00
FADD(3/2)	11	11	22(28)	0.83	0.51	1.34	11	10	21(27)	0.63	0.39	1.01	11	10	21(27)	0.63	0.37	1.00
GBC(4/3)	28	25	53(61)	4.18	0.37	4.55	35	34	69(77)	0.97	0.48	1.45	29	26	55(63)	0.8	0.2	1.00
Total	251	247	**498(602)**	1.18	1.02	**2.19**	287	258	**545(649)**	0.62	0.55	**1.17**	241	225	**466(570)**	0.54	0.46	**1.00**

• The runtime of MOTO-X is controlled by iterations and is hard to be precisely controlled to be the same as that of "Hybrid", so MOTO-X is controlled to run longer than "Hybrid" to confirm the effectiveness and optimization limit.
• The transistor count inside the parentheses of "Total" includes the transistors of the inverter for generating negative switch literals.

without the hybrid process ("Binary" and "Hybrid"). One can notice from Table I that "Hybrid" has obvious runtime decreasing (about 14.53%) compared to "Binary". Further insights is available in Figure 7. The "Linear" (which is the linear search mentioned in Figure 6, and is the same as the linear search in MiniTNtk [14]), binary search "Binary", and the hybrid process "Hybrid" are tested. The results show that in all cases, linear search leads to more searches. In contrast, binary search significantly reduces the searches, and the hybrid process further trims the searches for most testcases. These benefit the speed of the proposed algorithm, making it faster than the state-of-the-art work.

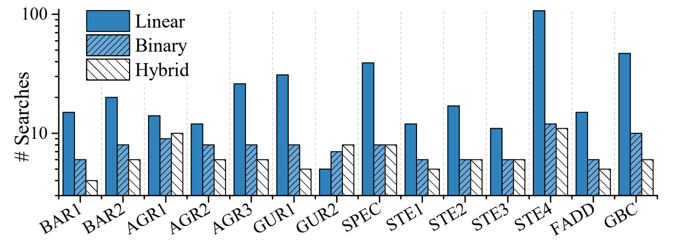


Fig. 7. The number of searches of different orders. The "Hybrid" has reduced maximum 90% and average 58% searches compared to "Linear".

Finally, we illustrate the insights from the experiments to show the effectiveness of the hybrid process, as depicted in in Figure 8. Due to space limitation, typical cases are listed. Figure 8 indicates the upper and the lower bound of the proposed algorithm when guided by the heuristic algorithm during the hybrid process, aligning with the design intention. It also indicates that the searched transistor count of the exact algorithm is well controlled by the estimated bounds. Note that for "STE1 PMOS", the searched count reaches a lower value than the bounds in the last iteration. This try is for optimality in case that the lower bound estimation is not perfect. The process has resulted in a runtime reduction for "Hybrid" compared to "Binary" for the relevant cases.

D. Discussion and Limitation

The hybrid algorithm incorporates an exact SAT-based multi-output logic algorithm. However, its lower bound of the transistor count searching is estimated by a heuristic algorithm,

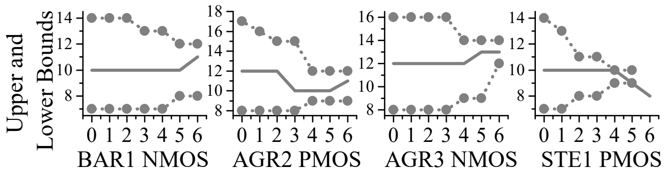


Fig. 8. The upper and lower bounds (dotted curves) updated by the hybrid process and the currently searched count (the solid curve) through time steps.

which may be incorrect. To maintain the exact property, once the searched transistor count reaches the lower bound and a solution is found, the search will continue by decreasing the lower bound by 1 until no solution can be found. In the experiments, only 4 test cases' searching reached the incorrect lower bounds, indicating that most bound estimates are reasonable. It's important to note that we have set a timeout feature (5k-10k seconds for each iteration) for practicality. This may affect the optimality of the exact algorithm, but it can still provide a solution that is optimized enough based on the experiments.

We also discuss the limitations of this work. For cases larger than those in the experiments, the SAT-based generation in the hybrid process can have high computing complexity, which may not produce a valid result within the time limit. However, due to the nature of the hybrid optimization, the result from the heuristic algorithm can still be obtained and outputted as the final result. On the other hand, since our work focuses on optimizing for standard cell libraries, other than very large cases, the sizes of the logic functions are usually lightweight and can be efficiently optimized by the proposed algorithm.

VIII. CONCLUSIONS

In conclusion, this work proposes a hybrid exact and heuristic transistor network optimization algorithm for multi-output logic CMOS circuits. The proposed algorithm not only contains the first exact algorithm supporting multi-output logic by SAT formulations, but also adopts the hybrid process with a heuristic algorithm to guide the exact algorithm for efficiency. Experimental results show the proposed hybrid algorithm has better transistor count and runtime, compared with both the state-of-the-art work for multi-output logic and the proposed algorithm without the hybrid process.

REFERENCES

[1] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentelli, “Sequential Circuit Design Using Synthesis and Optimization,” *IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pp. 328–333, 1992.

[2] M. C. Golumbic, A. Mintz, and U. Rotics, “An Improvement on the Complexity of Factoring Read-Once Boolean Functions,” *Discrete Applied Mathematics*, vol. 156, no. 10, p. 1633–1636, 2008.

[3] M. G. A. Martins, L. Rosa, A. B. Rasmussen, R. P. Ribas, and A. I. Reis, “Boolean Factoring with Multi-Objective Goals,” *IEEE International Conference on Computer Design*, pp. 229–234, 2010.

[4] M. G. A. Martins, V. Callegaro, L. Machado, R. P. Ribas, and A. I. Reis, “Functional composition paradigm and applications,” *International Workshop on Logic & Synthesis*, 2012.

[5] J. Zhu and M. Abd-El-Barr, “On the optimization of mos circuits,” *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 40, no. 6, pp. 412–422, 1993.

[6] V. N. Possani, R. S. de Souza, J. S. Domingues, L. V. Agostini, F. S. Marques, and L. S. da Rosa, “Optimizing Transistor Networks Using A Graph-Based Technique,” *Analog Integrated Circuits and Signal Processing*, vol. 73, no. 3, pp. 841–850, 2012.

[7] V. N. Possani, V. Callegaro, A. I. Reis, R. P. Ribas, F. de Souza Marques, and L. S. da Rosa, “Graph-Based Transistor Network Generation Method for Supergate Design,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 2, pp. 692–705, 2016.

[8] D. Kagaris, “MOTO-X: A Multiple-Output Transistor-Level Synthesis CAD Tool,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 114–127, 2016.

[9] L. S. da Rosa Junior, F. S. Marques, T. M. G. Cardoso, R. P. Ribas, S. S. Sapatnekar, and A. I. Reis, “Fast Disjoint Transistor Networks from BDDs,” *Symposium on Integrated Circuits and Systems Design*, p. 137–142, 2006.

[10] L. S. da Rosa, F. R. Schneider, R. P. Ribas, and A. I. Reis, “Switch Level Optimization of Digital CMOS Gate Networks,” *International Symposium on Quality Electronic Design*, pp. 324–329, 2009.

[11] L. S. da Rosa, A. I. Reis, R. P. Ribas, F. d. S. Marques, and F. R. Schneider, “A Comparative Study of CMOS Gates with Minimum Transistor Stacks,” *Conference on Integrated Circuits and Systems Design*, p. 93–98, 2007.

[12] D. Kagaris and T. Haniotakis, “A Methodology for Transistor-Efficient Supergate Design,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 4, pp. 488–492, 2007.

[13] V. N. Possani, V. Callegaro, A. I. Reis, R. P. Ribas, F. d. S. Marques, and L. S. da Rosa Junior, “Efficient Transistor-Level Design of CMOS Gates,” *ACM International Conference on Great Lakes Symposium on VLSI*, p. 191–196, 2013.

[14] W. Xiao, S. Han, Y. Yang, S. Yang, C. Zheng, J. Chen, T. Liang, L. Li, and W. Qian, “MiniTNtk: An Exact Synthesis-based Method for Minimizing Transistor Network,” *IEEE/ACM International Conference on Computer Aided Design*, pp. 01–09, 2023.

[15] C. Bolchini, G. Buonanno, D. Sciuto, and R. Stefanelli, “A New Switching-Level Approach to Multiple-Output Functions Synthesis,” *International Conference on VLSI Design*, pp. 125–129, 1995.

[16] G. Buonanno, D. Sciuto, and R. Stefanelli, “Innovative Structures for CMOS Combinational Gates Synthesis,” *IEEE Transactions on Computers*, vol. 43, no. 4, pp. 385–399, 1994.

[17] Gurobi Optimizer, https://www.gurobi.com/, 2023.