On Structural Bias of Two-Level Multi-Output Minimization

Petr Fišer[0000−0001−5306−6343] and Jan Schmidt[0000−0003−3221−7370]

Czech Technical University in Prague
Faculty of Information Technology
Department of Digital Design
fiserp@fit.cvut.cz, schmidt@fit.cvut.cz

Abstract. Two-level logic minimization is a well-established process that efficiently minimizes functions described in the Sum-of-Products (SOP) form. Current two-level minimizes, like Espresso and Boom, produce optimum or near-optimum results regarding the SOP size. However, there could be an issue when multi-level logic synthesis follows the two-level optimization. In this paper, we show that a two-level minimization conducted in the standard way significantly worsens the result of the subsequent multi-level synthesis, especially for sparsely-specified multiple-output functions, regardless of the optimization process and the tool used. In conclusion, we propose using a single-output two-level minimization for incompletely specified functions. In this way, the obtained circuits are sometimes multiple times smaller, while no significant deterioration has been observed for any other functions.

Keywords: logic synthesis · two-level minimization · structural bias · incompletely specified functions

1 Introduction

When the source to logic synthesis is a tabular description of a function (truth table, set of terms, the Berkeley PLA format), the well-established synthesis process is to run two-level minimization, e.g., Espresso [1] and then proceed with multi-level optimization and technology mapping. In the case of completely specified functions, running the two-level minimization *explicitly* is typically not even needed since multi-level synthesis tools, like SIS [12], ABC [7], and commercial tools, can do the job themselves. However, in the case of incompletely specified functions (e.g., where values of only some minterms are specified), preprocessing the function by a two-level minimizer is often necessary. For example, when reading an incompletely specified PLA, ABC substitutes the don't care values with zeroes, which loses many opportunities for efficient optimization. In general, ABC cannot deal with external don't cares [5].

On the other hand, SIS [12] is able to read incompletely specified PLAs and the don't cares are recognized. In particular, these external don't cares are added to observability don't cares [5] in the `full_simplify` command. However,

this procedure is rather time-consuming and hardly applicable to functions with many input variables.

Thus, when one wants to efficiently utilize don't cares in multi-level synthesis by, e.g., ABC, the two-level minimization should be run first, to produce a minimized *completely specified* function from an incompletely specified one, hoping that the don't cares would be assigned a value efficiently. A multi-level optimization and technology mapping is then run on this completely specified function.

This procedure is believed to yield "reasonable" results. Well, it does, for almost completely specified functions. However, we have found a big issue with *sparsely-specified functions*, i.e. functions where only few minterms have the values specified and the majority of minterms are don't cares. We have observed that the *multiple-output* two-level minimization of such functions introduces an unwanted structural bias [4], significantly affecting the subsequent multi-level synthesis and optimization.

The observed affection is not negligible – results almost three times as big as those generated by an "unconventional" way are produced. In particular, even though the multiple-output two-level minimization produces results having much fewer terms than the single-output one, the effect of the subsequent multi-level synthesis is the opposite.

In this paper, we document that for incompletely specified multi-output functions, running *any* multiple-output two-level minimization prior to the multi-level synthesis is not a good option. Running the *single-output* minimization instead helps significantly. This observation holds for all subsequent multi-level synthesis processes we have experimented with.

Moreover, we will show a very strange phenomenon appearing when synthesizing "larger" *random* sparsely-specified functions. It appears that there are two classes of such functions – the "easy" and "difficult" ones to synthesize, while being of the same size and nature.

The contributions of this paper can be summarized as follows. We show that:

- Doing multi-output minimization of incompletely specified random functions is not a good choice if a multi-level circuit synthesis is supposed to be run afterward.
- We show that there is even no difference in the multi-level synthesis tool and algorithm used when used after the two-level minimization – multi-output two-level minimization always does harm
- We illustrate a strange behavior for some classes of functions – easy and hard-to-synthesize functions of the same size and nature are identified.

The paper is structured as follows: Section 2 presents the motivation behind the research. Some of the related works are presented in Section 3. The experimental observations are presented in Section 4. A discussion of the most striking results is presented in Section 5. Finally, the conclusions are drawn in Section 6.

2 The Story Behind

When designing direct implementations of neural networks, i.e., where the network is implemented as a purely combinational circuit [8], we have run into serious problems. In particular, the neural network we wanted to synthesize into hardware was described as an incompletely specified function having relatively many (100) inputs, relatively many (100) outputs, and tenths of thousands of specified minterms. This function was described in a PLA format [9]. Note that despite the large size of the description, this function is still sparsely specified. In order to efficiently exploit the don't cares, we tried to run Espresso [1] to optimize the PLA. However, it took an extremely long time to run, and finally crashed. Thus, we have decided to sacrifice the optimality to some extent and split the function by outputs into two PLAs having 100 inputs and 50 outputs. Thus, the power of the multi-output two-level minimization cannot be fully exploited in this case. The split still did not suffice, so we continued. After all, when arriving at 25 outputs, the minimization has successfully finished. But still, the run times were very high (several days). Thus, we have decided to continue with the splitting, ending up with single-output minimization. The run times were obviously smaller.

To evaluate how much harm the splitting did, we have compared the results obtained by the application of splittings by different numbers of outputs. One observation was expected: *the more splitting was performed, the more terms and literals the resulting PLAs had*, since the multi-output minimization cannot be fully exploited in the case of splitting. However, the second observation was rather surprising: *the effects of the splitting on the multi-level optimization were exactly the opposite – the more output variables the functions submitted to Espresso had, the more gates had the final networks.* Moreover, this behavior was systematic for all the examples we have tried. After all, we have concluded that when the target technology is not a PLA, *a single-output minimization is the best option in the case of sparsely-specified functions.*

To illustrate the aforementioned observations, we have synthesized a neural network from the MNIST benchmark [3]. The neural network had three layers, each having 100 binary inputs and 100 binary outputs, with 60,000 specified minterms. Functions of different layers (L2, L3, and their concatenation L2.L3) were synthesized by Espresso and then by ABC, using a script mapping the circuit into arbitrary 2-level gates:

```
&st; &synch2; &if -m -a -K 2; &mfs -W 10;
&st; &dch; &if -m -a -K 2; &mfs -W 10.
```

The script was repeated 20-times.

The results are shown in Table 1. The columns represent the number of variables in the minimized functions. I.e., "1" represents a single-output minimization, while "100" represents group minimization of the original, non-split function. The rows indicate the respective functions. In each cell, the first value represents the number of distinct terms of the SOP obtained by Espresso, and the second value represents the number of 2-input gates after ABC synthesis.

Table 1. Splitting results

Layers		Outputs 1	2	4	5	10	20	25	100
L2	PLA terms	11,097	11,003	10,960	10,929	10,802	10,460	10,321	N/A
	2-input gates	33,810	36,394	41,635	43,606	53,974	71,531	73,371	
L3	PLA terms	6,409	6,326	6,202	6,165	5,884	4,317	5,635	N/A
	2-input gates	18,647	20,121	22,861	24,496	29,847	29,181	41,757	
L2.L3	PLA terms	17,309	17,142	17,005	16,932	16,270	15,511	15,129	N/A
	2-input gates	57,708	61,564	70,450	75,290	92,895	131,518	140,360	

For better illustration, we show the results for the most complex function (the L2.L3) in a graph, see Fig. 1.

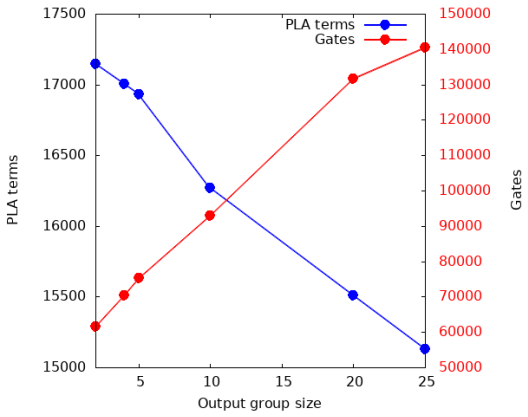

Fig. 1. The influence of group minimization on the number of PLA terms and the resulting number of 2-input gates after synthesis by ABC

It can be clearly seen that when the number of outputs minimized together increases, the number of PLA terms decreases (up to some minor exceptions, which can be attributed to random fluctuations). However, the number of gates after multi-level synthesis *consistently* increases. Therefore, these preliminary results indicate that running a single-output minimization prior to the multi-level optimization is the best option. This phenomenon will be further explored in this paper.

Note that the exercised functions mostly exhibit a random nature, so they substantially differ from standard benchmarks. However, such functions are becoming ever more important in practice, especially because of the advent of the direct implementation of neural networks [8].

3 Related Work

Needless to say, there are many means of handling incompletely specified functions. Available two-level minimizes [1], [6] indeed do take advantage of don't cares present (implicitly or explicitly) in the input PLA and use them to minimize (typically) the number of resulting product terms, as well as for generating group implicants, i.e. implicants of multiple output functions.

In multi-level synthesis, there are algorithms exploiting external don't cares – typically, they are appended to observability don't cares and used in Boolean optimization [11]. Other ways of handling incompletely specified functions in multi-level synthesis are described in [2]. Unfortunately, these algorithms are implemented in SIS [12] only (in the time-expensive `full_simplify` command) or are not available at all. The most state-of-the-art optimization tool, ABC [7], does not support external don't cares at all.

Sassao addressed the problem of "hard to optimize" functions in [10]. However, this concerned two-level minimization only, without any relation to multi-level synthesis.

As a result, we are not aware of any novel study on using don't cares in multi-level synthesis, when an incompletely specified function is described by a two-level (SOP) form. Therefore, we are bound to use a two-level minimizer prior to the multi-level optimization. In this paper, we show the issues caused by this.

4 Experimental Results

Having a clue that a single-output two-level minimization could yield better results for some "random" incompletely specified functions, we have performed exhaustive experimental work in order to find out *when* does it yield better results. We will restrict ourselves to exploring randomly generated functions (PLAs) where there is a given number (ratio) of *minterms* specified, as this scenario most closely resembles functions obtained from the direct implementation of neural networks [8].

In particular, we have generated random functions (PLAs, the `fr` type [1]) with a given number of inputs, outputs, and specified minterms. The output values of the minterms were also completely specified. Thus, the don't cares were given by the minterms missing in the PLA description. The distribution of ones and zeroes in both the AND plane and the OR plane was uniform, i.e., 50:50.

4.1 Multi-output vs. Single-output Minimization by Espresso

In the first experiment, these functions were processed in two ways:

1. Multi-output minimization by Espresso + synthesis into 2-input gates by ABC, as described in Section 2,

2. Single-output minimization by Espresso (using the `-Dso` option) + synthesis into 2-input gates by ABC, as described in Section 2.

In Figure 2, there are results for functions having 10 input and 2 output variables, with varying numbers of specified minterms, from 50 to 1024 (i.e., completely specified functions are at the rightmost side of the graph). For each number of minterms, 20 random functions were generated.

The graph shows the ratios of resulting 2-input gates, between the two processes. Thus, the values above 1 (the red line) are in favor of the second process. We can see that most of the values in the graph are above 1. Thus, the single-output minimization wins on average here, even with two output variables only.

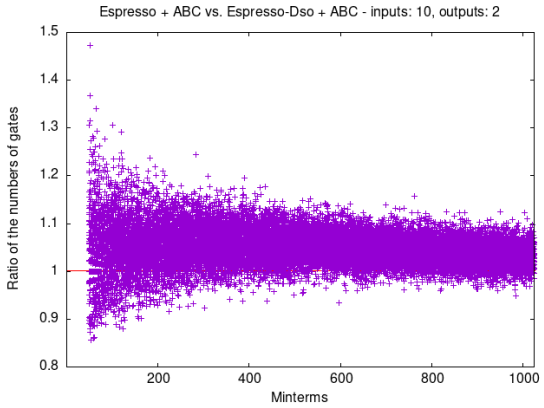

Fig. 2. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis by ABC, functions with 10 inputs and 2 outputs, 20 random functions for each number of specified minterms

Now, let's proceed with increasing the number of outputs. The results are shown in Figure 3. The number of outputs was varied from 3 to 15. We see an interesting trend here – when increasing the number of outputs, there starts to appear a "bump" at 20-30% of specified minterms, where the multi-output minimization results are the worst. Next, for more outputs, all the values are above one, thus the single-output minimization generally wins.

Moreover, we can observe a strange "dynamic" behavior. When the number of outputs becomes 10, we can see a see that there starts to appear a "gap" between the points in the graphs. This gap increases with increasing the number of outputs. This indicates that there seems to be a class of functions that are "hard" and "very hard" for this synthesis scenario. The strange thing is that there are no functions in between. When increasing the number of outputs further, the gap disappears, in favor of those "very hard" functions. This phenomenon is even emphasized when we increase the number of inputs, as shown in Figure 4, for 11 inputs.

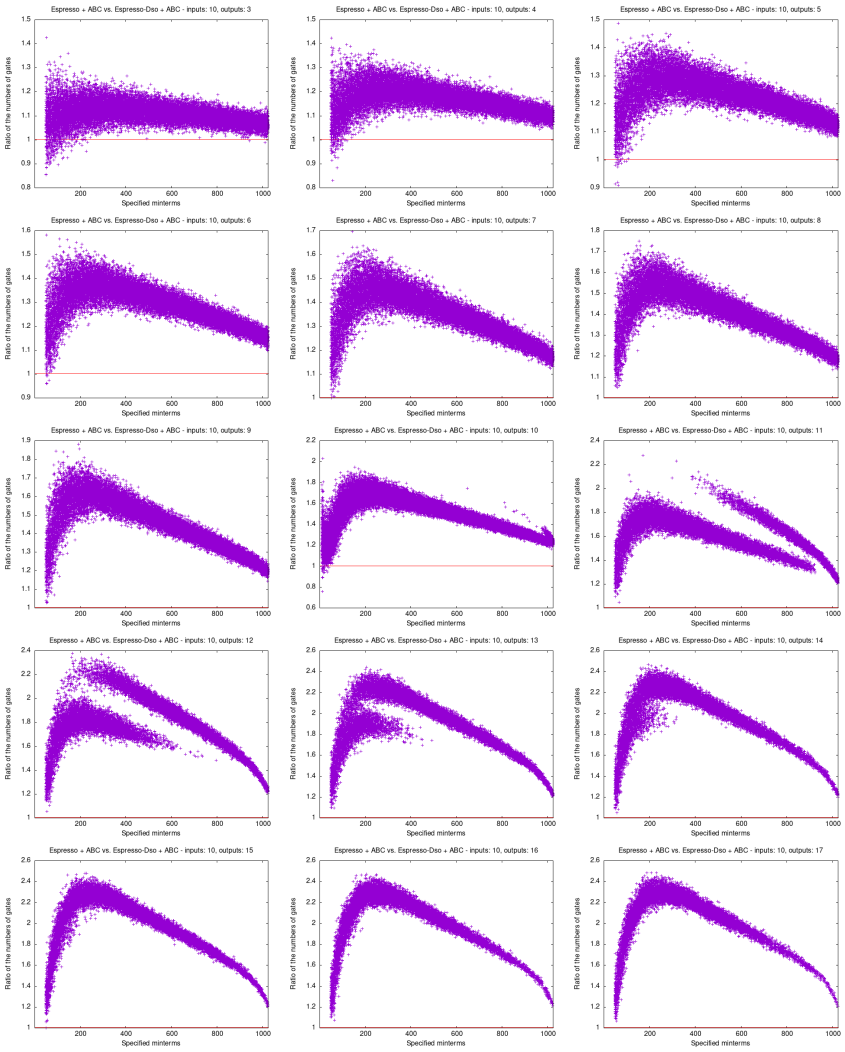

Fig. 3. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis by ABC, functions with 10 inputs and 3-17 outputs, 20 random functions for each number of specified minterms

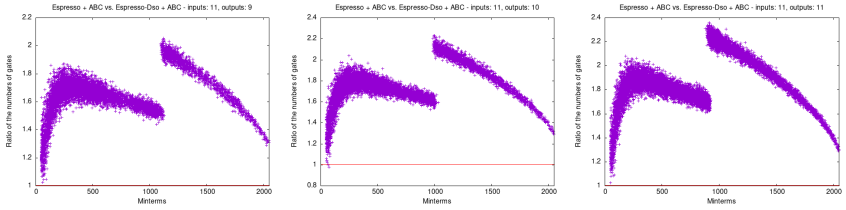

Fig. 4. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis by ABC, functions with 11 inputs and 9-11 outputs, 20 random functions for each number of specified minterms

4.2 Multi-output vs. Single-output Minimization by Boom

Now we may ask whether the phenomenon observed above is caused by Espresso or whether it is a general problem of multi-output two-level minimization. Thus, we have used Boom [6] instead of Espresso. Again, we have compared multi-output and single-output minimization processes followed by ABC synthesis.

An example of the obtained results can be seen in Fig. 5, for a function having 10 inputs and 10 outputs. We can see a very similar figure here - there is that "bump" at ca. 25% specified terms. The difference from Espresso is that there are no such "gaps" that can be observed in Figs. 3 and 4 – even when the number of input variables increases, the behavior looks the same.

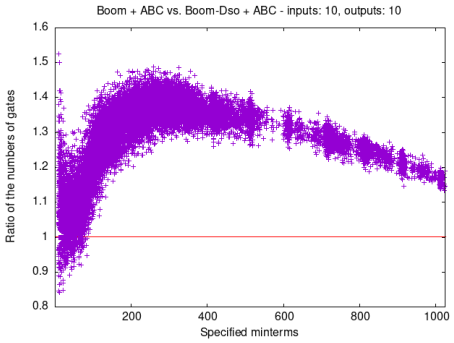

Fig. 5. Comparison of multiple- and single-output minimization by Boom run prior to the synthesis by ABC, functions with 10 inputs and 10 outputs, 40 random functions for each number of specified minterms

4.3 Boom vs. Espresso

Now let us compare the results obtained by Boom and Espresso, for both multiple-output and single-output minimization followed by ABC optimization and syn-

thesis into 2-input gates. The results are shown in Figure 6. All four combinations are shown there.

It can be seen that in the case of a multi-output minimization run for both Boom and Espresso, Boom outperforms Espresso in almost all cases (the upper-left figure). The same holds for the multi-output minimization by Espresso compared to a single-output minimization by Boom (the bottom-left figure). On the other hand, the upper-right figure shows that the single-output minimization done by Espresso wins over the multi-output minimization done by Boom.

The most important figure is the bottom-right one, where both Espresso and Boom were run in the single-output mode. Here we can see no difference between these two processes – there is no winner. The minor differences can be attributed to random fluctuations. Thus, this indicates that *the single-output two-level minimization is better than the multi-output one, no matter what two-level minimizer is used.* Note that Espresso and Boom conduct the minimization in very different ways, actually in opposite ones (Espresso goes in a bottom-up way, while Boom goes in a top-down way and by completely different means).

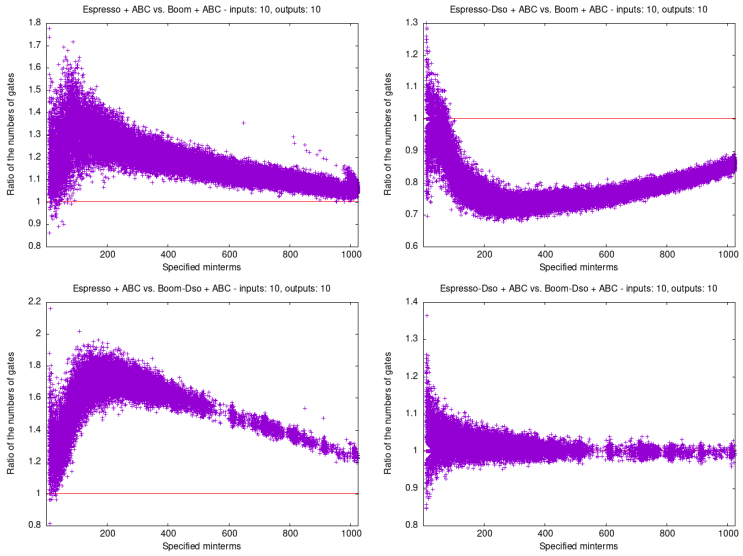

Fig. 6. Comparison of multi-output minimization by Espresso with Boom (multiple-output and single-output minimization) run prior to the synthesis by ABC, functions with 10 inputs and 10 outputs, 40 random functions for each number of specified minterms

4.4 Espresso vs. SIS

The other way of exploiting external don't cares is using SIS and processing them in the multi-level synthesis directly, without the two-level minimization.

Thus, we have compared the results of running Espresso (both single- and multi-output minimization) with SIS script `script_rugged` iterated 20-times. Both were followed by the ABC mapping script shown in Sec. 2. The results are shown in Figure 7, for a function of 6 inputs and 15 outputs. For larger functions, the SIS script took too much time to finish for all the numbers of minterms.

We can see that in the case of multi-output Espresso minimization, the same behavior (the "bump") can be observed, and the results are overall inferior to those of SIS. In the case of single-output minimization, the results are comparable but still in favor of SIS. However, this might be expected because of the strength of the Boolean optimization in SIS.

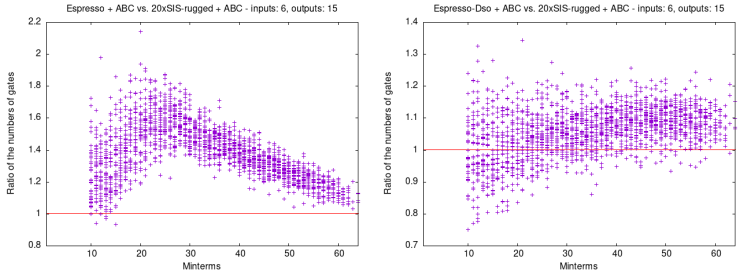

Fig. 7. Comparison with the SIS script `script_rugged` and the multi-output and single-output minimization by Espresso run prior to the synthesis by ABC, functions with 6 inputs and 15 outputs, 40 random functions for each number of specified minterms

4.5 Commercial Tools

The final question we may ask is whether ABC is not responsible for the deficiency, as it was run in all experiments. Thus, we have tried to use commercial tools, particularly Vivado 2023.1 (Artix-7 chip) and Quartus 22.1 (Cyclone V chip). In particular, the functions were processed by Espresso (both the single- and multiple-output minimization), the resulting PLA was converted to VHDL and submitted to Vivado and Quartus synthesis. The numbers of resulting 6-LUTs (for Vivado) and ALMs (for Quartus) were counted. The results are shown in Figures 8 and 9.

We can see that the behavior is rather similar to ABC. As for Vivado, we can observe that "bump" again, even though it is in the region with more (ca. 90%) don't cares. When the functions become more specified, the effect of two-level minimization completely disappears. In the case of Quartus, the results are similar or worse – multi-output minimization always causes harm. The behavior for functions with 8 input and output variables is ... just strange.

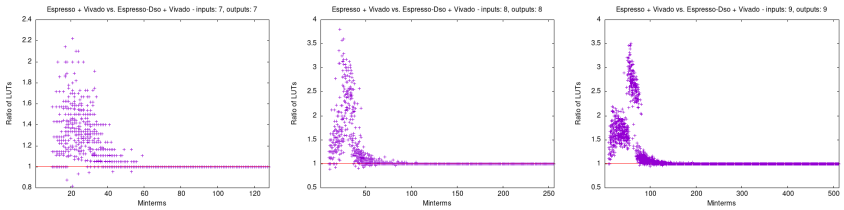

Fig. 8. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis by Vivado, functions with 7-9 inputs and outputs, 20 random functions for each number of specified minterms

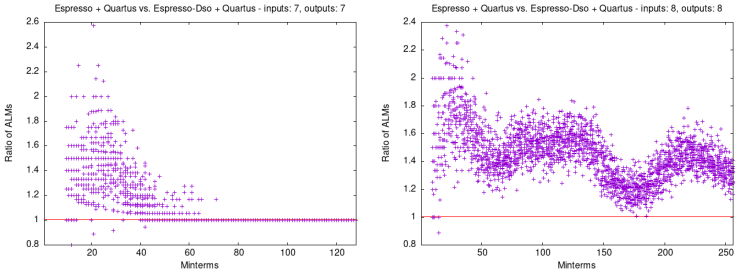

Fig. 9. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis by Quartus, functions with 7-8 inputs and outputs, 20 random functions for each number of specified minterms

5 Discussion on the Espresso Results

We have done extensive experimental work to prove that the observed kinds of behavior are not just random outliers. The behavior is consistent over randomly generated functions with different numbers of input and output variables. The summary of some observations is shown in Table 2, just for the multiple- and single-output minimization by Espresso, followed by the ABC synthesis. We can classify the results obtained from processing functions of different numbers of input (rows) and output variables (columns) by the cases described above. This is:

- Class A - no difference between the multiple- and single-output minimization is observed
- Class B - the expected cases – single-output minimization is generally better
- Class C - the usual cases – we can see that "bump" there
- Class D - the strange cases – for some function sizes, there is the "gap", indicating there are some hard and very hard to synthesize functions
- Class E - the bad cases – the multi-output minimization badly fails in all cases

The missing values were not calculated because of excessive computation power demands, but we think the results are representative enough.

Table 2. Summary of the Espresso results. A = no difference, B = expected cases, C = usual cases, D = The strange cases, E = when things go really bad

Inputs/Outputs	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
5	A	A	B	B	B	B	B	B	B	B	B	C	C	C	C	C	C	C	C
6	A	A	B	B	B	B	C	C	C	C	C	C	C	C	C	C	C	C	C
7	A	B	B	B	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
8	A	B	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
9	A	B	C	C	C	C	C	D	D	D	D	E	E	E	E	E	E	E	E
10	B	C	C	C	C	C	C	C	D	D	D	D	D	E	E	E	E	E	E
11	B	C	C	D	D	D	D	D	D	D									
12	B	D	D	D	D	D	D	D	D	D	D								

6 Conclusion

We have shown experimentally that the structural bias introduced by the multiple-output two-level minimization of incompletely specified functions is the cause of the inefficiency of the subsequent multi-level synthesis. In the case of *sparsely-specified functions*, the two-level minimization (done either by Espresso or Boom) is just not able to exploit the don't cares efficiently when the target implementation is a multi-level network, no matter what multi-level optimization process

follows. We have shown that the best option is to use a single-output two-level minimization since the multiple-output two-level minimization only spoils the result. This observation can be generalized for all randomly generated incompletely generated functions, which actually resemble the state-of-the-art applications of logic synthesis – the synthesis of neural networks implemented as combinational circuits.

Acknowledgment

Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

References

1. Brayton, R.K., Sangiovanni-Vincentelli, A.L., McMullen, C.T., Hachtel, G.D.: Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers, Norwell, MA, USA (1984)
2. Chang, K.H., Bertacco, V., Markov, I.L., Mishchenko, A.: Logic synthesis and circuit customization using extensive external don't-cares. ACM Trans. Des. Autom. Electron. Syst. **15**(3) (jun 2010). https://doi.org/10.1145/1754405.1754411, https://doi.org/10.1145/1754405.1754411
3. Deng, L.: The MNIST database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine **29**(6), 141–142 (2012). https://doi.org/10.1109/MSP.2012.2211477
4. Fiser, P., Schmidt, J., Balcarek, J.: Sources of bias in EDA tools and its influence. In: IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems. pp. 258–261 (April 2014)
5. Hachtel, G.D., Somenzi, F.: Logic Synthesis and Verification Algorithms. Kluwer Academic Publishers, Norwell, MA, USA (1996)
6. Hlavicka, J., Fiser, P.: BOOM – a heuristic boolean minimizer. In: IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281). pp. 439–442 (2001). https://doi.org/10.1109/ICCAD.2001.968667
7. Mishchenko, A., et al.: ABC: A system for sequential synthesis and verification (2012), http://www.eecs.berkeley.edu/~alanmi/abc
8. Nazemi, M., Pasandi, G., Pedram, M.: Energy-efficient, low-latency realization of neural networks through boolean logic minimization. In: 24th Asia and South Pacific Design Automation Conference. pp. 274–279 (Jan 2019)
9. Rudell, R., Sangiovanni-Vincentelli, A.: Multiple-valued minimization for PLA optimization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **6**(5), 727–750 (September 1987)
10. Sasao, T., Butler, J.: Worst and best irredundant sum-of-products expressions. IEEE Transactions on Computers **50**(9), 935–948 (2001). https://doi.org/10.1109/12.954508
11. Savoj, H., Brayton, R.K.: The use of observability and external don't cares for the simplification of multi-level networks. In: Proceedings of the 27th ACM/IEEE

Design Automation Conference. p. 297–301. DAC '90, Association for Computing Machinery, New York, NY, USA (1991). https://doi.org/10.1145/123186.123280, https://doi.org/10.1145/123186.123280

12. Sentovich, E., Singh, K., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: SIS: a system for sequential circuit synthesis. Tech. Rep. UCB/ERL M92/41, EECS Department, University of California, Berkeley (1992)