Structure Analysis in Boolean Functional Synthesis

Ziv Avissar[1] and Dror Fried[1]

The Open University of Israel, Israel

Abstract. Boolean Functional Synthesis (BFnS) is the problem of synthesizing a Boolean function from a Boolean specification that describes a relation between input and output variables. Due to the many applications of BFnS, such as in circuit design, QBF solving, and reactive synthesis, efforts are continuously made to explore and better study BFnS. In this work we deepen our understanding of BFnS by analyzing underlying graph structures of BFnS instances. This is motivated by a chain of works on SAT instances, where analysis of graph features, such as graph modularity, is used to explore the performance of SAT solvers on industrial SAT instances. We first show that unlike instances that are random, industrial BFnS instances admit high modularity, even more than is common in SAT. Observing that, we construct a novel BFnS random instance generator with controlled modularity, which we use to study the effect of modularity on performance of state-of-the-art BFnS solvers. Finally we white-box these solvers to determine how the structure of generated instances changes iteratively through the solving process. Our findings indicate that instances modularity has a direct effect on the various solvers performance and their behavior.

Boolean Functional Synthesis (BFnS) is the problem of extracting a boolean function from a given Boolean specification that describes a relation between input and output variables. Due to its clean formulation, BFnS has found applications in many areas, among which circuit design, QBF solving, reactive synthesis, and automated program synthesis. As such, study of BFnS has evolved over the last decade and various research groups have suggested different approaches and tools to battle BFnS (e.g. [11, 2, 6, 9, 8]). Even so, BFnS still remains a very hard problem to solve at a large scale, which is not surprising as it is a computational complexity challenging problem, specifically Π_2^P-hard. Moreover, it is very likely that not all BFnS instances even admit a polynomial size solution [1].

One of the means to battle BFnS is to reason on the structure of the given specification. There are already a few stratagems for solving BFnS such as factorization [9] and sequential decomposition [5] that utilize decomposition of the specification into clauses. A question to ask, however, is whether there is a more inherent structure of BFnS specifications that can be related to solutions runtime by the various BFnS solvers. This challenge is already met for the problem of satisfiability (SAT) that also consists of input as a Boolean formula. Indeed, discerning structure from a given Boolean formula has been explored in SAT in recent years [3]. Specifically, it was shown that unlike random Boolean formulas, real-world originated (or "industrial") Boolean formulas, admit certain graph properties when represented as graphs. This resulted in a line of research that utilized these properties to understand SAT problems in a deeper way [3, 4].

In this work we elevate the study of structure analysis of SAT to BFnS, thus making the first step in structure analysis of BFnS. Our research questions are whether BFnS real-world instances have designated graph properties, distinguished from random BFnS instances, and whether such graph properties, if exist, affect the behavior of BFnS solvers. An affirmative answer to these questions can open the door to a more thorough study that seeks to improve BFnS tools performance by exploiting these graph properties.

To answer these questions, we use graph representations of Boolean formulas that are natural for BFnS. As BFnS is related to SAT, we examine the Variable Incidence Graph (VIG) and Clause to Variable Incidence Graph (CVIG) that are used for SAT structure analysis in [3, 4]. We also examine the Input Consensus Graph (ICG) representation that is used in the process of a BFnS solver called Back and Forth (BnF) [5]. The graph property that we analyze in these graph representations is graph modularity, related to the graph community structure, and used in SAT structure analysis. For benchmarks we use both random generated BFnS instances and so-called *real-world* or *industrial* BFnS instances, originating from real problems, thus having some semantic interpretations.

We first show that, unlike random instances, industrial instances admit very high graph modularity for their VIG. Typically, graph modularity ranges within the interval [0,1] and is defined high within [0.3,0.7]. For SAT benchmarks the average is 0.15 for random instances and around 0.7 for industrial SAT instances [3]. For BFnS industrial instances the average modularity is 0.75 - above 0.5 for nearly all benchmarks and as high as 0.9 for some. The reason for that could be that in real world BFnS problems - some variables occur much more than others. These findings positively answer our first research question and serve as a promising start. While modularity distinctions were also noted for the other graph representations, we decided from this point to focus more on the popular VIG graph and its graph modularity (henceforth called VIG modularity).

Our next step is to find how the VIG modularity of BFnS instances affects solvers performance. To meet this challenge, however, we need to find benchmarks with spanned VIG modularity over the [0,1] range. Note that we cannot use real-world benchmarks as we already showed that their VIG modularity is mostly focused in a small interval. As such, we construct a random BFnS instance generator called **ModQBF** with which we can control the VIG modularity of the generated instances. Our generator, based on a similar random SAT instance generator [7], can specifically produce BFnS benchmarks of VIG modularity similar to those of industrial instances. Thus as a second use, **ModQBF** can produce random instances that are one-step closer to real world ones.

Using **ModQBF** , we deepen our analysis and check how various BFnS solvers perform on our randomly generated BFnS instances with controlled modularity. The solvers that we use are current state-of-the-art solvers: CADET [10], BnF [5], BFSS [1] and Manthan [8]. Our results positively answer our second research question as we show that the average run-time of all solvers is indeed affected by the change of modularity. Specifically we show that as the VIG modularity increases, the average run-time of all solvers tends to decrease, whether moderately as in CADET and BFSS, or more drastically as BnF and Manthan.

Finally, we construct a monitoring method to better understand the relation between the structure of the BFnS benchmarks and the solvers performance. For that, we observe that all the solvers in mention work by iterative transformations and additions to the given formula as a part of the execution process. We white-boxed the solvers to find the precise places, called *iteration points*, in the solvers algorithms in which each change of the processed formula occurs. Then we extracted the structure of the formula in these iteration points for analysis. This gives us an abundance of data out of which we point out several observations of interest that relate the benchmarks VIG modularity and BFnS solvers performance. This solidifies the point that these metrics are not transcendental qualities chosen arbitrarily, but rather valid and relevant metrics in the field of BFnS for classifying problems and analyzing solvers. The results also point out that the use of these metrics still has much more yet to unveil.

References

1. Akshay, S., Chakraborty, S., Goel, S., Kulal, S., Shah, S.: What's hard about boolean functional synthesis? In: Computer Aided Verification - 30th International Conference, CAV, Proceedings, Part I (2018)
2. Akshay, S., Chakraborty, S., John, A.K., Shah, S.: Towards parallel boolean functional synthesis. In: Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS, Proceedings, Part I (2017)
3. Ansótegui, C., Bonet, M.L., Giráldez-Cru, J., Levy, J., Simon, L.: Community structure in industrial SAT instances. J. Artif. Intell. Res. **66**, 443–472 (2019)
4. Ansótegui, C., Bonet, M.L., Levy, J.: On the structure of industrial SAT instances. In: Principles and Practice of Constraint Programming - CP, 15th International Conference, Proceedings (2009)
5. Chakraborty, S., Fried, D., Tabajara, L.M., Vardi, M.Y.: Functional synthesis via input-output separation. In: 2018 Formal Methods in Computer Aided Design, FMCAD (2018)
6. Fried, D., Tabajara, L.M., Vardi, M.Y.: Bdd-based boolean functional synthesis. In: Computer Aided Verification - 28th International Conference, CAV, Proceedings, Part II (2016)
7. Giráldez-Cru, J., Levy, J.: A modularity-based random SAT instances generator. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI (2015)
8. Golia, P., Roy, S., Meel, K.S.: Manthan: A data-driven approach for boolean function synthesis. In: Computer Aided Verification - 32nd International Conference, CAV, Proceedings, Part II (2020)
9. John, A.K., Shah, S., Chakraborty, S., Trivedi, A., Akshay, S.: Skolem functions for factored formulas. In: Formal Methods in Computer-Aided Design, FMCAD (2015)
10. Rabe, M.N., Tentrup, L., Rasmussen, C., Seshia, S.A.: Understanding and extending incremental determinization for 2QBF. In: Computer Aided Verification - 30th International Conference, CAV, Proceedings, Part II (2018)
11. Tabajara, L.M., Vardi, M.Y.: Factored boolean functional synthesis. In: 2017 Formal Methods in Computer Aided Design, FMCAD (2017)