Genetic Algorithm for Functionally-Equivalent and Structurally-Divergent Benchmark Generation

David S. Koblah
Florida Institute for National Security
University of Florida
Gainesville, FL, USA
Email: dkoblah@ufl.edu

Rabin Acharya
Florida Institute for National Security
University of Florida
Gainesville, FL, USA
Email: rabin.acharya@ufl.edu

Domenic Forte
Florida Institute for National Security
University of Florida
Gainesville, FL, USA
Email: dforte@ece.ufl.edu

***Abstract*—A trending topic in nearly every field these days, including electronic design automation (EDA) is machine learning (ML) and artificial intelligence (AI). However, to properly train and evaluate such approaches, especially when deep learning is involved, a very large number of benchmarks is necessary. This paper delves into the application of information theory for circuit benchmark assessment; we utilize evolutionary algorithms for exhaustive exploration of the design search space. Building upon existing research utilizing genetic algorithms and Prolog-based binary trees, our approach focuses on genetic algorithms, employing crossover, mutation, and selection on populations generated from input benchmarks. While the initial work halves the binary tree structure of smaller designs, we enhance its scalability in terms of benchmark complexity and applicability. We focus on two design goals: size optimization and diverse benchmark generation. The fitness function used to assess the feasibility of a result is based on the information theory concepts of mutual and normalized mutual information. Its successful implementation also creates opportunities for other optimization objectives, such as security-oriented ones.**

***Keywords*—digital circuit; binary decision diagram; genetic algorithm; information theory**

I. Introduction

Artificial intelligence (AI) and machine learning (ML) techniques can play an essential role in electronic design automation (EDA) [1]. Benefits include but are not limited to guiding design tradeoffs, educating users on best design practices, detecting bugs and mitigating them, recommending the best test strategies and much more. The state-of-the-art today is deep learning which has been able to solve some of the most persistent problems in speech and face recognition, object detection, etc. However, vast amounts of quality data are required to deliver effective AI/ML models and solutions. The EDA community is plagued by a lack of benchmarks with most researchers still relying on common sets such as the ISCAS ('85,'89, '99) and OpenCores benchmarks. Although these have varying degrees of complexity, these designs do not accurately capture the entire space of today's IC industry and will not result in the most robust AI models. In this paper, we employ information theory for for circuit benchmark assessment and utilize evolutionary algorithms for exhaustive exploration of the design search space for optimization.

The concept of information theory deals with measuring and analyzing information in signals or data. It has applications in telecommunications, data compression, cryptography, and machine learning. Information theory especially finds some applicability to the assessment of circuit benchmarks. Entropy-based equations with input values from a design can be used to achieve specific objectives, optimization, or otherwise [2], [3]. However, without an effective means of exploring the full range of possible options, information theory loses the opportunity to find the most suitable solution. In this work, evolutionary computations are used to exhaustively explore the design search space. Evolutionary computations are a family of algorithms for global optimization inspired by the Darwinian concept of biological evolution [4]. They are suitable for optimization tasks that require approximate solutions with minimum possible overhead. Examples include Swarm Intelligence [5], Cultural Algorithms [6] and Genetic Algorithms [7].

Our work supplements the earlier publication that uses evolutionary computation to explore the search space while applying entropy calculations for similarity measures [2]. Like our predecessors, we specifically focus on genetic programming, which allows us to utilize the concepts of crossover, mutation, and selection on synthetic populations generated from an input benchmark. We provide two design goals; one focuses on structural optimization while the other attempts to minimize decision diagram sizing. The structural optimization goal has the same motivation as described in [8]. However, in this paper, we are able to maintain the input benchmark's function while also increasing structural diversity and optimizing other objectives. The current implementation works exclusively on combinational benchmarks but can be applied to both single and multiple output benchmarks. This is facilitated by the use of Binary Decision Diagrams (BDDs) instead of the binary multiplexers used in [2]. A BDD is a directed acyclic graph used to represent a Boolean function [9]. For genetic programming, the fitness (suitability) function is created using the information theory concepts of mutual and normalized mutual information, as seen in [10]. An optimum fitness value maximizes the normalized mutual information between the original design and the proposed solution benchmark.

Our contributions are summarized as follows:

DISTRIBUTION STATEMENT A. Approved for public release: distribution is unlimited.

1) **Sizing Optimization**: Exploration of diverse forms of sizing optimization for the provided benchmarks, accommodating a range of design objectives.
2) **Functionally-Divergent Benchmarks**: Introduction of benchmarks exhibiting functional divergence to enhance their suitability for comprehensive testing purposes.
3) **Modular Algorithm Pipeline**: Provision of a flexible algorithm pipeline, featuring modularity and adaptability through interchangeable fitness functions.

The rest of the paper reads as follows: Section II discusses the information theory and other concepts applied in this paper. Section III highlights previous work combining the concepts applied in our proposed work. Section IV explains our method, showing the two-stage approach to achieving a benchmark result. Section V merges the experimental setup with results on selected benchmarks. Section VI concludes the paper and provides directions for future work.

II. Background

A. Information Theory and Shannon's Entropy

Information theory under applied mathematics and electrical engineering simply involves quantifying information. One of the key concepts in information theory is *entropy*, as introduced by Claude Shannon [11]. Entropy is a measure of uncertainty or randomness associated with a random variable or a set of data, and within the context of information theory, it represents the average amount of surprise or unpredictability in a message [12], [13]. Shannon's entropy formula is expressed as

$$H(X) = -\sum_{i=1}^{n} P(x_i) \cdot \log_2(P(x_i)) \tag{1}$$

where $H(X)$ is the entropy of the random variable X, n is the number of possible outcomes and $P(x_i)$ is the probability of outcome x_i. The formula calculates the expected value of the information content of a message, measured in bits. When the probability distribution is more uniform across outcomes, the entropy is higher, indicating higher uncertainty. Conversely, if the distribution is more concentrated, the entropy is lower.

In the context of circuit design and optimization, entropy-based measures can be applied to assess the randomness or complexity of Boolean functions [2], [14]. These measures are often used in genetic algorithms and other search space optimization techniques to guide the search for more efficient designs. By leveraging entropy-based fitness functions, a designer can explore and synthesize circuits that balance complexity and functionality.

B. Evolutionary Algorithms

Evolutionary Algorithms are inspired by the process of Darwinian-based natural selection. They operate by evolving a population of candidate solutions over successive generations to find optimal or pareto solutions to a given problem [16]. The key components, including ones used in this paper are:

1) *Initialization*: A population of potential solutions is created, often randomly or using specific heuristics.
2) *Selection*: Solutions are evaluated based on a fitness function, and individuals with higher fitness are more likely to be selected for reproduction.
3) *Crossover*: Pairs of selected individuals exchange genetic information to create new solutions, mimicking the crossover of genetic material in biological reproduction.
4) *Mutation*: Random changes are introduced to some individuals, adding diversity to the population and expanding the solution search space.
5) *Termination*: The algorithm stops when a termination criterion is met, such as reaching a maximum number of generations or achieving a satisfactory solution.

Examples include *genetic algorithms* used in various optimization problems like job scheduling and financial modeling, and *differential evolution* widely used for numerical optimization problems in engineering design, parameter tuning, and robotics. These algorithms are especially suitable for complex, high-dimensional problems where traditional optimization approaches may struggle [17].

C. Binary Decision Diagrams

Binary Decision Diagrams (BDDs) are data structures used to represent and manipulate Boolean functions efficiently. They provide a compact and canonical form for expressing logical relationships in a binary tree structure [18]. Within digital circuit design, they are used for hardware verification and optimization. The key features of BDDs are:

- Nodes: Each node represents a Boolean variable.
- Edges: Directed edges connect nodes, indicating the variable's assignment (0 or 1).
- Terminal Nodes: The tree ends with terminal nodes representing the constant values 0 and 1.

The benefits of using BDDs include:

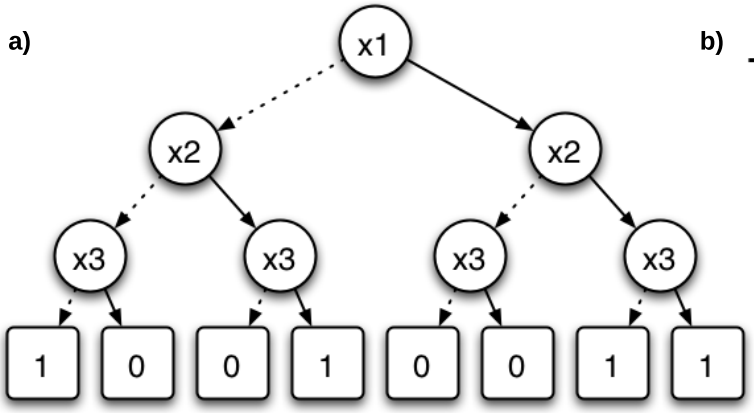
- Compact Representation: BDDs can represent large Boolean functions more efficiently than explicit truth tables.
- Canonical Form: BDDs have a unique, canonical form, ensuring consistency in representation.
- Efficient Operations: BDDs support efficient operations like conjunction, disjunction, and negation.

An example of a BDD and the logic function if represents can be found in Figure 1.

III. Literature Review

As mentioned in Section I, information theory has found utility in various data-reliant fields. However, for operations on digital circuits, the work we build on combines both information theory and evolutionary algorithms. The authors explore the application of pure Genetic Programming for synthesizing logic circuits using binary multiplexers, guided solely by a fitness function based on entropy [2]. The Shannon expansion [12] of a Boolean formula is identified as a reliable foundation for the evolutionary method. The paper suggests the use of Information Theory, particularly entropy-based measures like Mutual Information (MI) and Normalized

DISTRIBUTION STATEMENT A. Approved for public release: distribution is unlimited.



b)

x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Fig. 1. a) Binary Decision Diagram with corresponding b) truth table for $f = (\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$ from [15]

Mutual Information (NMI), in designing fitness functions for Boolean circuit synthesis. Three fitness functions are developed, highlighting the suitability of NMI as a fitness function.

Echavarria et al. present a novel methodology for Approximate Computing, specifically in the context of multi-output logic functions [19]. Unlike existing approaches that minimize each output function independently, our method considers the collective impact on all outputs, maximizing cross-function minimization potential. Integrated into a design space exploration technique, our approach provides a Pareto-set of designs, offering trade-offs between hardware costs and error. Experimental results demonstrate efficiency, achieving significant reductions in terms and literals with minimal inaccuracy. [19]

Another paper addresses challenges in Approximate Computing, focusing on accelerating error metric computations for logic approximation [20]. The proposed methods leverage structural information and estimate metrics for multi-output Boolean functions represented as BDDs. A greedy, bucket-based BDD minimization framework incorporates these error metric computations, producing Pareto-optimal solutions in terms of BDD size and multiple error metrics. Experimental results demonstrate speed improvements and approximated BDDs.

Finally, an article introduces a novel framework for synthetic circuit benchmark generation to address the limitations of using traditional benchmarks in hardware security research [8]. Leveraging principal component analysis (PCA) and linear optimization, the proposed framework produces divergent benchmarks with maximum structural variation. The framework allows user customization for desired features, enhancing the generation of challenging benchmarks for data-driven hardware security.

IV. Proposed Methodology

The approach outlined in this paper is structured into two main components. The initial part encompasses **pre-evolutionary operations** essential for obtaining the BDD that serves as the foundational structure for the framework. The subsequent phase builds on the DEAP Python package designed explicitly for genetic algorithm operations [21]. Further details regarding these are elaborated in the subsequent subsections.

A. Pre-Evolutionary Operations

The process begins with the input of a gate-level netlist, which is the best-suited representation of the logical connections between various gates in a digital circuit. The original netlist is then converted into a directed graph (digraph), maintaining the relationships and dependencies among different elements in the circuit. This digraph becomes a crucial data structure for the algorithm's operation, because its directional edges can show fanin and fanout information. The two actions described above are enabled by the circuitgraph Python package [22].

The next step involves the random generation of a population of individuals from the target digraph created above. These individuals collectively form a diverse set that mirrors the potential designs of the digital circuit. The algorithm then transforms these digraphs into BDDs, as seen in Fig. 2. As described in Section II-C, the BDD structure enhances efficiency in handling and evaluating the logic of the circuits. The evolutionary process unfolds through selection, crossover, and mutation operations on the population of BDDs. The conversion is facilitated by pyEDA, a Python library dedicated to EDA [23]. This open-source package offers several features, including symbolic Boolean algebra with diverse function representations like logic expressions, truth tables with "don't care" states, and BDDs.

1) Utilization of Binary Decision Diagrams: Our current implementation focuses specifically on combinational designs. The choice of Boolean function representations is crucial for manipulating the actual circuit designs and leveraging Boolean reasoning. As described above, various data structures are available for representing Boolean functions, including truth tables, Boolean formulas in the form of sums of products (SOP) and products of sums (POS), and Boolean networks. In our case, we opt for BDDs due to their Shannon

DISTRIBUTION STATEMENT A. Approved for public release: distribution is unlimited.

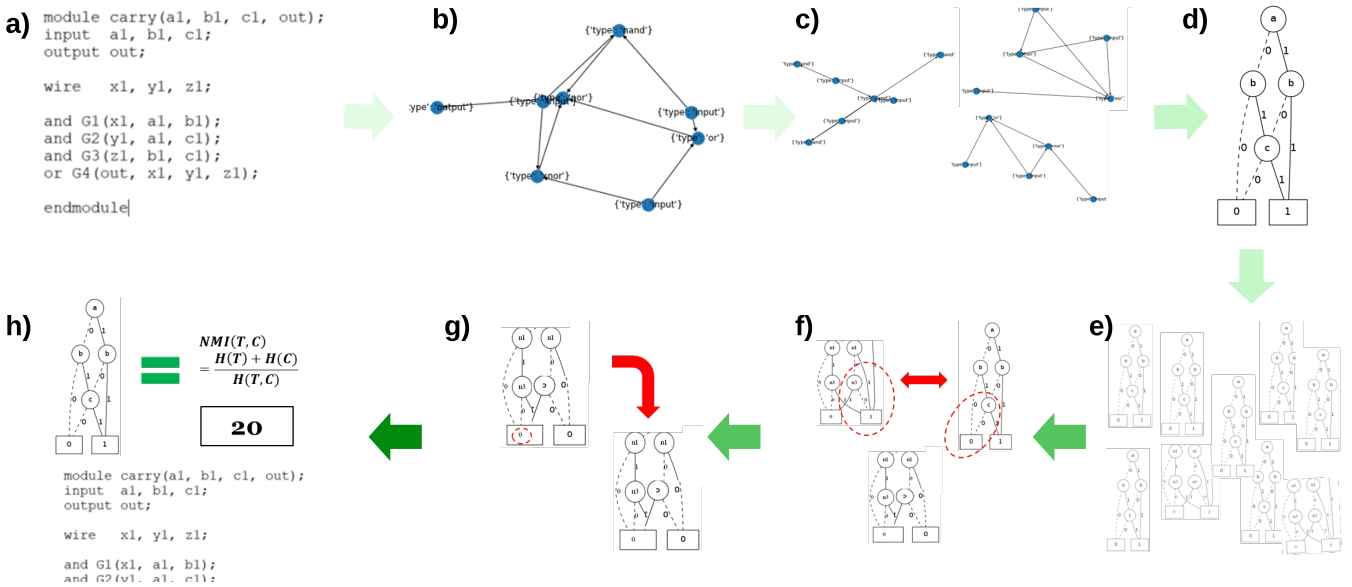


Fig. 2. Summary of framework for functionally-equivalent benchmark generation using genetic algorithm: a) Input of gate-level netlist; b) Conversion of original netlist into digraph; c) Random generation of population as digraphs d) Conversion of digraphs into BDD representation; e) Selection; f) Crossover; g) Mutation; h) Best-fit individual and benchmark generation.

```
toolbox = base.Toolbox()
toolbox.register("mate", tools.cxOnePoint)
toolbox.register("mutate",
                 tools.mutShuffleIndexes,
                 indpb=0.3)
toolbox.register("select", tools.selBest,
                 k=30)
```

Fig. 3. Excerpt of DEAP declarations required for evolutionary operations in Python environment. "mate" is the crossover function set to single point, "mutate" is set to a 30% likelihood of an attribute being mutated, while "select" indicates selection of the top 30 individuals per generation.

core factor tree structure, variable ordering restriction, and reduction rules, which make the representation canonical. BDDs facilitate easy and efficient logic operations while maintaining functionality, provided the variables are appropriately ordered [18]. The graph structure of BDDs proves highly suitable for applications involving graph-based problems, and their parallelizability allows for scalable processing of larger Boolean functions.

B. DEAP Programming for Evolution

DEAP is a framework that merges the full functionality of the Python programming language with a straightforward and streamlined core of transparent Evolutionary Computation components [21]. This combination is well-suited for the application of existing and novel ideas in Evolutionary Algorithms while maintaining ease of use through simple and explicit algorithms.

Selection involves identifying individuals with favorable traits based on the selected fitness function. This eliminates non-functional members of the generated population while providing the most fit individuals. The best 30 individuals are selected to proceed (see Fig. 3). The next step, *crossover*, combines genetic information from two parent individuals to produce offspring, simulating the recombination of genetic material in biological evolution. The DEAP framework provides various options for crossing, including two-point and single-point implementations. We empirically find that the latter favors our implementation.

Mutation introduces random changes to individual BDDs, adding diversity to the population. The function shuffles the attributes of the crossed over individual and returns the mutant. Since at this stage of the frame work, the bdd is manipulated as a sequence, mutation is enabled by randomly moving indexes. The *indpb* argument in Fig. 3 is the probability of each attribute to be moved.

The algorithm continuously refines the population through these evolutionary operations, aiming to improve the fitness of individuals over successive generations. The best-fit individual, representing an optimized design, is identified based on the defined fitness criteria. Additionally, a gate-level circuit is generated from this optimized design, providing a reference for evaluating and comparing the algorithm's performance.

C. The Ideal Fitness Function

The exploration of the relationship between the target Boolean function (T) and the circuit output (C) involves entropy-based metrics. To generate the effective fitness function, Conditional Entropy from Eqn. (2) and Joint Entropy from Eqn. (3) are required. Referring to [2], the conventional fitness function, driven solely by mutual information (MI), has exhibited suboptimal performance (Eqn. (4)). The paper

DISTRIBUTION STATEMENT A. Approved for public release: distribution is unlimited.

advocates for fitness functions rooted in Normalized Mutual Information (NMI) (Eqn. (5)).

$$H(T|C) = -\sum_{i=1}^{n}\sum_{j=1}^{n} p(t_i, c_j) \log_2 \frac{p(t_i, c_j)}{p(c_j)} \tag{2}$$

$$H(T, C) = -\sum_{i=1}^{n}\sum_{j=1}^{n} p(t_i, c_j) \log_2 p(t_i, c_j) \tag{3}$$

$$MI(T, C) = H(T) + H(C) - H(T, C) \tag{4}$$

$$NMI(T, C) = \frac{H(T) + H(C)}{H(T, C)} \tag{5}$$

This adjustment in the fitness function aims to enhance the efficacy of the genetic programming algorithm in synthesizing logic circuits. It creates a more regularized search space for exploration. Further information on the development of the NMI-based fitness function is detailed in [2].

V. Experiments and Results

The experimental setup serves the purpose of obtaining sizing solutions and diverse benchmarks. These benchmarks are crucial for testing and are sourced from various public datasets. All experiments are executed on the HiPerGator supercomputer at the University of Florida. The resources accessed are 4 CPU cores with 80 GB of memory. This is supplemented by a single GeForce 2080Ti GPU unit.

The results obtained from the experiments are detailed in Table I. The optimization process leads to sizing solutions, and in some instances, the original design is retained as intended (termination requirement). For designs with multiple outputs, we leverage the fan-in for each output to create a single function. In this iteration, the focus is on optimizing individual functions. If common terms appear among these functions, we achieve overall optimization. The fitness function selected for use is:

$$(\text{Length}(T) - \text{Hamming}(T, C)) \times (10 - \text{H}(T|C)) \tag{6}$$

It is based on conditional entropy in Eqn. (2) and uses the mentioned factor to suppress the reproduction of undesirable individuals [2].

A. Running Time

The running time of the experiments is directly influenced by the complexity of the given benchmark. Through empirical observations, it is noted that designs with more than 25 inputs posed substantial challenges in terms of processing, regardless of the computer resources utilized. A key focus for the next iteration is to parallelize operations on circuits of this nature and their respective BDDs. This technique will enable the generation of appropriate benchmarks and solutions within a reasonable timeframe. Furthermore, the parallelization process will speed up operations on current complex designs, such as the *int2float* benchmark in Table I.

B. Generation Count

Another notable result observed during the experimentation process is the relationship between generation counts and optimal population size. We find that these factors exhibited a direct correlation with the size of the original benchmark. Larger benchmarks necessitated higher population counts and increased generation numbers, expanding the search space. The experimentation begins with 100 generation counts and expanded to 22 iterations to enhance the likelihood of finding satisfactory solutions. Similarly, an initial population size of 100 is used, but adjustments are made based on the optimal number of designs obtained and their usability. Notably, an unnecessarily large population size is indicated by repeated fitness function values across selected individuals in generations. Determining the ideal population size is crucial for achieving substantial diversity in fitness results and design types, a factor that influences the production of benchmarks.

C. Benchmark Divergence

The fitness function results are employed as a measure of the extent of divergence in the context of divergence benchmarks. For instance, a fitness score of 18 indicates that the design is structurally more divergent than a design with a score of 19 while maintaining identical functionality to the original target function. Therefore, by establishing specific thresholds for particular designs, we can control the level of structural divergence we desire. Leveraging this approach enables the generation of multiple benchmarks, which proves valuable for various digital circuit experiments. Table II shows results on the same benchmarks used in Table I. We note that while ranges can be set for divergent benchmarks, further work is required to fine-tune requirements for results.

VI. Conclusion and Future Work

In this paper, we present the outcomes of experiments that yield viable results in the production of size-optimized designs using a genetic algorithm framework. Additionally, we successfully generate structurally divergent benchmarks employing the fitness function proposed. These benchmarks serve as valuable datasets for future testing and experimentation. While improving upon the previous work by incorporating BDDs and synthesizing multiple output functions, we encounter challenges related to access runtimes and limited optimization for these functions. We also note that our experiments exclusively focus on combinational designs.

For future work, we plan to integrate parallelization with enhanced genetic algorithm frameworks, providing support for sequential designs. Moreover, we aim to broaden our optimization objectives to explicitly include structural considerations and other design requirements for digital circuits.

VII. Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1651701 as well as the SMART Scholarship-for-Service Program from the Department of Defense.

DISTRIBUTION STATEMENT A. Approved for public release: distribution is unlimited.

!t

TABLE I
RESULTS FOR DESIGNS FROM VARIOUS SOURCES. OPTIMIUM SOLUTIONS ARE ONES THAT MATCH THE FUNCTIONALITY OF THE TARGET CIRCUIT.

Benchmark	Source	Inputs	Outputs	Average Running Time/s	Average Generations	Optimum Solution
test.v	[2]	3	1	100.33	2200	No
gpt.v	ChatGPT 3.5	3	3	166.59	983.333	Yes
c17.v	[24]	4	2	227.92	2200	No
z4ml.v	[25]	7	4	498.69	2200	No
int2float.v	[26]	11	7	31499.97	1257.571	Yes
two_bit_multiplier.v	[26]	4	4	213.08	2200	No

TABLE II
RESULTS FOR DIVERGENT DESIGNS FROM SOURCES IN TABLE I. THRESHOLD VALUE IS LARGEST FITNESS INTEGER VALUE THAT PRODUCES FUNCTIONAL BENCHMARK

Benchmark	Threshold Value	Time to Solution
test.v	17	2.70
gpt.v	19	13.49
c17.v	18	149.91
z4ml.v	18	209.82
int2float.v	19	28983.51
two_bit_multiplier.v	17	103.35

REFERENCES

[1] D. Koblah, R. Acharya, D. Capecci, O. Dizon-Paradis, S. Tajik, F. Ganji, D. Woodard, and D. Forte, "A survey and perspective on artificial intelligence for security-aware electronic design automation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, no. 2, pp. 1–57, 2023.

[2] A. H. Aguirre and C. A. Coello Coello, "Evolutionary synthesis of logic circuits using information theory," *Artificial Intelligence Review*, vol. 20, pp. 445–471, 2003.

[3] F. Xiong and M. M. Tanik, "An experiment on evolutionary design of combinational logic circuits using information theory," in *2011 Proceedings of IEEE Southeastcon*, 2011, pp. 379–383.

[4] D. Dumitrescu, B. Lazzerini, L. C. Jain, and A. Dumitrescu, *Evolutionary computation*. CRC press, 2000.

[5] J. Kennedy, "Swarm intelligence," in *Handbook of nature-inspired and innovative computing: integrating classical models with emerging technologies*. Springer, 2006, pp. 187–219.

[6] R. G. Reynolds, "An introduction to cultural algorithms," in *Proceedings of the 3rd annual conference on evolutionary programming, World Scientific Publishing*. World Scientific, 1994, pp. 131–139.

[7] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[8] S. Amir and D. Forte, "Eigencircuit: Divergent synthetic benchmark generation for hardware security using pca and linear programming," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5207–5219, 2022.

[9] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a bdd package," in *Proceedings of the 27th ACM/IEEE design automation conference*, 1991, pp. 40–45.

[10] T. M. Cover, J. A. Thomas *et al.*, "Entropy, relative entropy and mutual information," *Elements of information theory*, vol. 2, no. 1, pp. 12–13, 1991.

[11] E. M. Guizzo, "The essential message: Claude shannon and the making of information theory," Ph.D. dissertation, Massachusetts Institute of Technology, 2003.

[12] R. B. Ash, *Information theory*. Courier Corporation, 2012.

[13] F. M. Reza, *An introduction to information theory*. Courier Corporation, 1994.

[14] T. Sasamal, H. Gaur, A. Singh, and A. Mohan, "Reversible circuit synthesis using evolutionary algorithms," *Design and Testing of Reversible Logic*, pp. 115–128, 2020.

[15] M. McGhee, "Binary decision diagrams," Jun 2018. [Online]. Available: https://medium.com/@michmac202/binary-decision-diagrams-e6b7c76392e

[16] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary computation*, vol. 1, no. 1, pp. 1–23, 1993.

[17] D. Dasgupta and Z. Michalewicz, *Evolutionary algorithms in engineering applications*. Springer Science & Business Media, 2013.

[18] Akers, "Binary decision diagrams," *IEEE Transactions on computers*, vol. 100, no. 6, pp. 509–516, 1978.

[19] J. Echavarria, S. Wildermann, and J. Teich, "Design space exploration of multi-output logic function approximations," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[20] A. Wendler and O. Keszocze, "A fast bdd minimization framework for approximate computing," in *2020 Design, Automation & Test in Europe Conference Exhibition (DATE)*, 2020, pp. 1372–1377.

[21] F.-A. Fortin, F.-M. De Rainville, M.-A. G. Gardner, M. Parizeau, and C. Gagné, "Deap: Evolutionary algorithms made easy," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 2171–2175, 2012.

[22] J. Sweeney, R. Purdy, R. D. Blanton, and L. Pileggi, "Circuitgraph: A python package for boolean circuits," *Journal of Open Source Software*, vol. 5, no. 56, p. 2646, 2020.

[23] C. Drake, "Pyeda: Data structures and algorithms for electronic design automation," in *Proc. 14th Python in science conference (SciPy)*, 2015, pp. 26–31.

[24] H. Fujiwara, "Iscas'85 benchmarks: Special session on atpg and fault simulation."

[25] C.-C. Tsai and M. Marek-Sadowska, "Multilevel logic synthesis for arithmetic functions," in *Proceedings of the 33rd annual Design Automation Conference*, 1996, pp. 242–247.

[26] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, no. CONF, 2015.

DISTRIBUTION STATEMENT A. Approved for public release: distribution is unlimited.