General Boolean Function Benchmark Suite

Roman Kalkreuth[⋆], Zdeněk Vašíček[†], Jakub Husa[†], Diederick Vermetten[‡], Furong Ye[‡] and Thomas Bäck[‡]

[⋆]Sorbonne University (Paris, France), [†]Brno University of Technology (Brno, Czech Republic), [‡]Leiden University (Leiden, The Netherlands)

Abstract

- Just over a decade ago, the first review on benchmarking in Genetic Programming (GP) stated that GP needs better benchmarks [McDermott et al., 2012].
- Several benchmark suites in major GP problem domains have been proposed over time.
- A diverse and accessible benchmark suite for logic synthesis is still missing [McDermott et al., 2022].
- We present the first GP benchmark suite that covers different types of Boolean functions.

Logic Synthesis in Genetic Programming

- Logic synthesis (LS) in GP can be considered a black-box and optimization problem domain.
- LS by means of GP refers to the synthesis of expressions that match the input-output mapping of Boolean functions.
- LS has played a major role in the application scope of GP research throughout its history.

General Motivation

- A general benchmark suite for LS is still missing in the field of GP.
- Promotion of LS to maintain a vivid application scope in GP.
- Cultivating diversity and accessibility in this problem domain.

Main Objectives and Properties

- The philosophy behind our proposed benchmark suite respects the following objectives and properties: **Generalization**, **Accessibility** and **Scalability**.
- A generalized approach to benchmarking in LS is achieved by covering a broad spectrum of types of Boolean functions.
- We make use of the scaling property of Boolean functions by proposing benchmarks of the same type with different bit lengths of the respective inputs.
- Scaling up the bit length increases the difficulty of the proposed benchmarks but maintains the similarity.

Problem Selection

- We thoughtfully selected Boolean functions from seven popular categories: **arithmetic**, **transmission**, **comparison**, **counting**, **mixed**, **parity** and **cryptography**.
- We selected high dimensional multiple-output functions that remarkably differ from the overused single-output parity and multiplexer benchmarks.
- The synthesis of cryptographic Boolean functions depicts a comparatively young and interesting field of application in LS and it is therefore natural to propose these functions as benchmarks.

Table 1. Abbreviations, names, and types of the selected benchmark functions.

Identifier	Name	Category
add	Adder with carry	Arithmetic
mul	Multiplier	Arithmetic
dec	One-hot decoder	Transmission
enc	One-hot encoder	Transmission
icomp	Identity Comparator	Comparison
mcomp	Magnitude Comparator	Comparison
count	Ones' counter circuit	Counting
alu	Arithmetic Logic Unit	Mixed
epar	Parity Even	Parity
bent	Bent	Cryptography
bal	Balanced	Cryptography
res	Resilient	Cryptography
mask	Masking	Cryptography

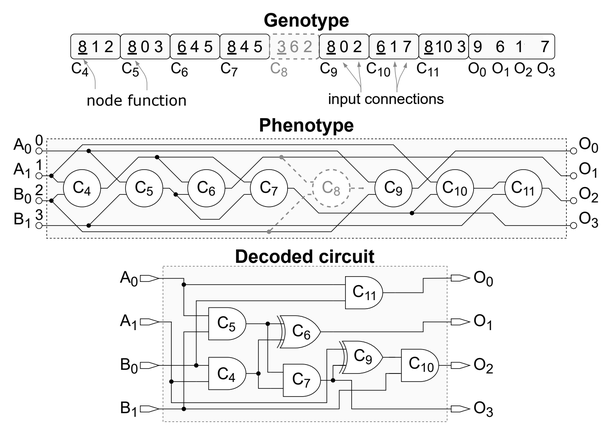
Interfaces and Resources

- Data files for the benchmarks are available in several formats and publicly accessible in our GitHub repository.
- Interfaces for C++, Java and Python can also be found in our repository.

https://github.com/boolean-function-benchmarks

Baseline results

- Baseline results have been obtained with Cartesian Genetic Programming (CGP) [Miller and Thomson, 2000].
- CGP is a graph-based representation model for GP that can be used for automated circuit design (see figure below).
- For our experiments, we used standard CGP with the $(1 + \lambda)$ evolutionary strategy.



CGP encodes an acyclic and directed graph that can be used to represent a candidate circuit.

Table 2. Baseline results obtained in our experiments. For the evaluation of each benchmark, we performed 100 runs and measured the number of fitness evaluations. A limit of 10^8 fitness evaluations was defined in each run.

Problem			Search Performance Evaluation					
Identifier	Inputs count	Output count	Average	Standard deviation	Q1	Median	Q3	Success rate
add4	9	5	1,176,250	1,241,222	559,810	864,099	1,244,870	100%
add6	13	7	4,354,846	3,425,693	2,098,854	3,358,271	4,883,583	100%
add8	17	9	15,118,552	12,166,477	6,549,025	11,308,473	18,063,144	100%
mul3	6	6	439,106	337,382	229,458	317,891	524,669	100%
mul4	8	8	37,074,573	16,213,144	23,682,377	36,943,707	47,053,177	100%
mul5	10	10	-	-	-	-	-	0%
dec4	4	2	11,520	11,199	4,722	8,326	12,079	100%
dec8	8	3	656,502	612,189	277,681	449,462	826,220	100%
dec16	16	4	16,657,577	9,659,375	9,822,953	13,514,673	20,891,128	100%
enc8	3	8	22,895	12,751	13,383	20,015	29,372	100%
enc16	4	16	133,268	55,065	90,761	125,830	163,579	100%
enc32	5	32	6,048,000	5,253,472	1,669,344	4645,978	8,446,886	100%
icomp5	5	30	298,378	253,280	148,066	231,331	351,607	100%
icomp7	7	63	1,476,305	771,096	906,376	1,318,098	1,812,708	100%
icomp9	9	108	5,472,300	2,475,782	3,761,209	5,102,597	6,401,704	100%
mcomp4	8	3	1,418,028	951,804	733,138	1,033,987	2,051,997	100%
mcomp5	10	3	2,898,336	1,923,573	1,593,730	2,489,199	3,416,328	100%
mcomp6	12	3	7,975,125	5,560,237	4,075,013	6,523,368	10,284,085	100%
count4	4	3	44,626	42,380	19,567	30,017	47,265	100%
count6	6	4	853,994	890,100	368,770	649,935	1,108,540	100%
count8	8	4	8,506,276	8,519,358	2,992,012	6,490,877	9,768,313	100%
count10	10	5	29,674,777	17,411,768	16,582,947	26,897,332	40,192,421	86%
alu4	11	5	11,342,677	5,995,435	6,961,372	10,437,724	14,009,597	100%
alu6	15	7	45,341,485	20,465,133	26,869,723	42,071,437	59,961,675	93%
alu8	19	9	72,811,646	16,498,376	63,702,578	74,942,768	85,688,638	23%
epar8	8	1	2,215,101	1,436,094	1,147,611	1,877,298	2,709,974	100%
epar9	9	1	4,232,008	2,935,317	2,130,972	3,375,780	5,159,903	100%
epar10	10	1	8,155,620	6,950,750	3,518,499	5,847,038	10,574,672	100%
epar11	11	1	11,869,051	9,964,340	6,123,499	8,183,564	13,465,052	100%
bent8	8	1	1,831	4,736	514	881	1,504	100%
bent12	12	1	3,262	2,764	1,543	2,514	4,045	100%
bent16	16	1	8,128	8,161	2,867	5,219	10,067	100%
bal8	8	1	103,807	124,303	27,635	68,280	143,775	100%
bal12	12	1	394,556	438,565	101,052	218,533	543,450	100%
res8	8	1	16,712	40,763	3,849	7,062	13,405	100%
res12	12	1	170,153	178,829	39,825	100,651	248,155	100%
mask8	8	1	3,406	3,694	1,536	2,481	3,919	100%
mask12	12	1	59,812	60,904	23,446	38,928	75,591	100%

References

[McDermott et al., 2022] McDermott, J., Kronberger, G., Orzechowski, P., Vanneschi, L., Manzoni, L., Kalkreuth, R., and Castelli, M. (2022). Genetic programming benchmarks: Looking back and looking forward. *SIGEVOlution*, 15(3).

[McDermott et al., 2012] McDermott, J., White, D. R., Luke, S., Manzoni, L., Castelli, M., Vanneschi, L., Jaskowski, W., Krawiec, K., Harper, R., De Jong, K., and O'Reilly, U.-M. (2012). Genetic programming needs better benchmarks. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, page 791–798, New York, NY, USA. Association for Computing Machinery.

[Miller and Thomson, 2000] Miller, J. F. and Thomson, P. (2000). Cartesian genetic programming. In Poli, R., Banzhaf, W., Langdon, W. B., Miller, J. F., Nordin, P., and Fogarty, T. C., editors, *Genetic Programming, European Conference, Edinburgh, Scotland, UK, April 15-16, 2000, Proceedings*, volume 1802 of *Lecture Notes in Computer Science*, pages 121–132. Springer.

Roman.Kalkreuth@lip6.fr