Evolutionary Search of Binary Orthogonal Arrays

Luca Mariot*, Stjepan Picek†, Domagoj Jakobovic‡, Alberto Leporati*

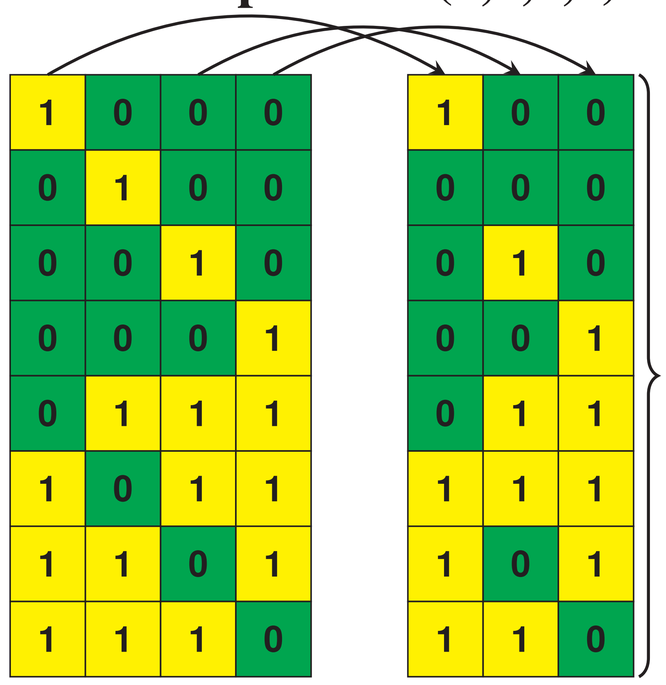
*University of Milano-Bicocca, †Delft University of Technology, ‡University of Zagreb

{luca.mariot, alberto.leporati}@unimib.it, s.picek@tudelft.nl, domagoj.jakobovic@fer.hr

Orthogonal Arrays

- An (N, k, t, λ) binary *Orthogonal Array* (OA) is a $N \times k$ binary matrix A such that in each $N \times t$ submatrix each binary t-uple occurs exactly λ times.
- OA have applications in *cryptography*, *statistics* and *coding theory*, and their construction is a difficult combinatorial optimization problem
- Existing *algebraic constructions* do not provide for a great variety of OA
- **Goal**: Construct binary OA with *Genetic Algorithms* (GA) and *Genetic Programming* (GP)

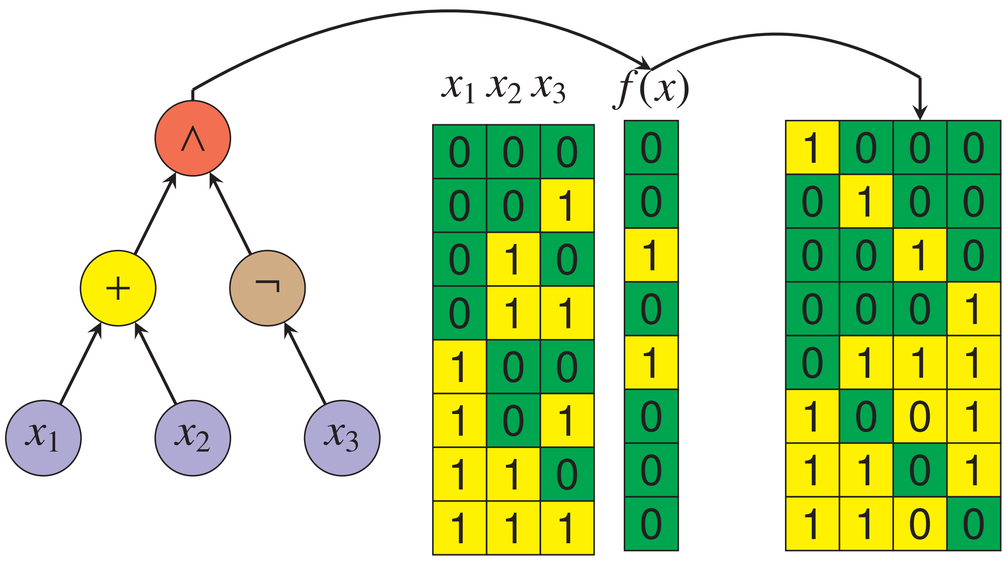
Example: OA (8,4,3,1)



⇒ Each 3-bit vector $(x_1, x_2, x_3) \in \{0,1\}^3$ appears once in the submatrix with columns 1, 3, 4

Solution Encoding

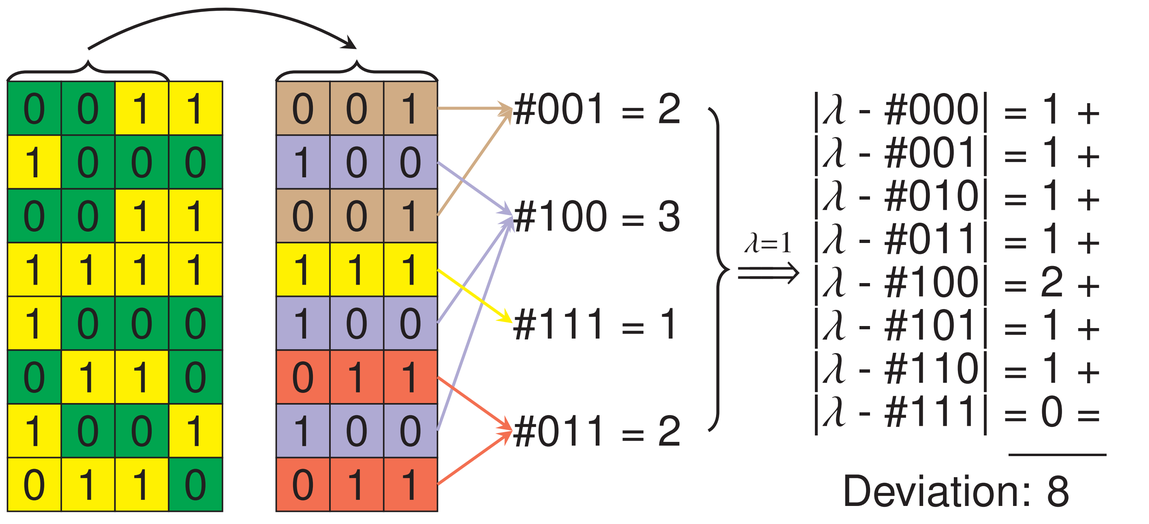
- Each column of a binary matrix is the 2^n-bit *truth table* of a n-variable *Boolean function*
- In the case of GP, the truth table is synthesized from the Boolean tree of the individual



- Genetic operators (e.g., crossover) are applied column-wise over individuals in the population

Fitness Function

- **Idea:** *minimize* in each $N \times t$ submatrix the number of occurrences of each t-uple deviating from λ



- **Fitness function:** L^p distance between vector $(\lambda, \cdots, \lambda)$ and the vector of deviations for each submatrix

$$fit_p(A) = \sum_{S \text{ Submatrix}} \left(\sum_{x \in \{0,1\}^t} |\lambda - \#x|^p \right)^{\frac{1}{p}}$$

Experimental Settings

- Problem instances/Orthogonal Array parameters considered in our experiments:

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
n	3	3	3	3	3	4	4	5	5	6
N	8	8	8	8	16	16	16	32	32	64
k	4	4	5	7	8	8	15	16	31	32
t	2	3	2	2	2	3	2	3	2	3
λ	2	1	2	2	4	2	4	4	8	8

- **GP function set:** AND, OR, XOR, XNOR (2 arguments), NOT (1 argument), IF (3 arguments)
- **Population sizes:** 500 (GP), 50 (GA)
- **Common parameters:** 500 000 fitness evaluations for each run, 30 independent runs for each problem instance

Results

- **Main finding:** GP outperforms by far GA at converging to an optimal solution over the considered problem instances

	GA					GP				
Exp.	min	avg	std	max	time (s)	min	avg	std	max	time (s)
I_1	0	0	0	0	< 1	0	0	0	0	< 1
I_2	0	0	0	0	< 1	0	0	0	0	< 1
I_3	0	0	0	0	< 1	0	0	0	0	< 1
I_4	0	0.533	1.38	4	7	0	0	0	0	1
I_5	0	2.333	1.75	6	38	0	0	0	0	1
I_6	0	39.96	10.9	57.41	110	0	0.565	3.09	16.97	13
I_7	52	65.4	6.41	80	147	0	0.533	2.03	8	48
I_8	1174	1266	43.4	1349	1995	0	83.72	41.5	135.8	1212
I_9	654	684	14.5	714	1 125	0	32	13.9	64	692
I_{10}	-	-	-	-	-	18812	19159	116	19355	15308

Conclusions

- Evolutionary Algorithms represent a *practical method* for the design of OA, capable of competing with algebraic constructions
- GP is a better metaheuristic than GA at handling combinatorial design problems, since we observed a similar difference in performance also in the evolution of *Orthogonal Latin Squares* in a previous work (GECCO 2017)