University of Milano-Bicocca
Department of Informatics, Systems and Communications

Evolutionary Optimization of Combinatorial Designs

Luca Mariot

luca.mariot@unimib.it

Trieste – January 15, 2020

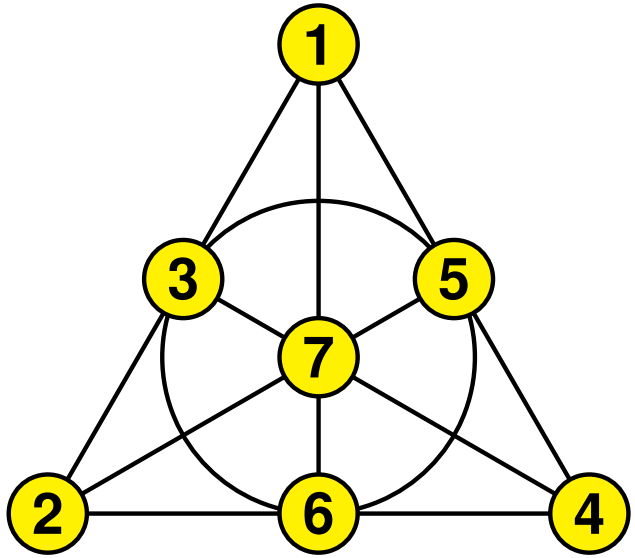
Introduction to Combinatorial Designs

What is a Combinatorial Design (CD)?

- A collection $\mathcal{A}$ of subsets (or **blocks**) of a finite set X satisfying particular **balancedness** properties

- Example: the **Fano Plane**

$$X = \{1,2,3,4,5,6,7\}$$
$$\mathcal{A} = \{123, 145, 167, 246,$$
$$257, 347, 356\}$$

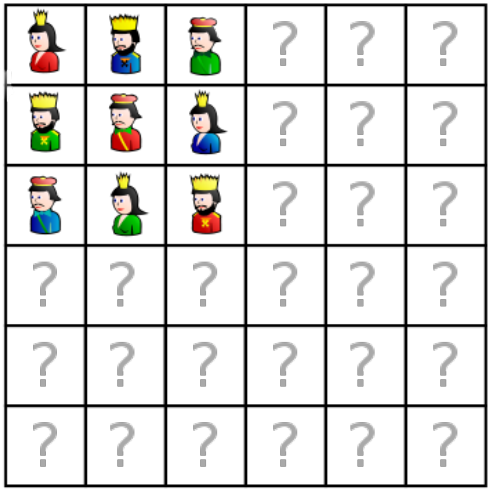


- Each block in $\mathcal{A}$ has 3 elements and each pair of distinct points in X occurs in exactly 1 block
- $\Rightarrow (7,3,1)$-BIBD (**Balanced Incomplete Block Design**)

Euler's 36 Officers Problem

« *A very curious question [...] revolves around arranging 36 officers to be drawn from 6 different ranks and also from 6 different regiments so that they are ranged in a square so that in each line (both horizontal and vertical) there are 6 officers of different ranks and different regiments.* »

L. Euler, *Sur une nouvelle espèce de quarrés magiques*, 1782



Latin Squares

Definition

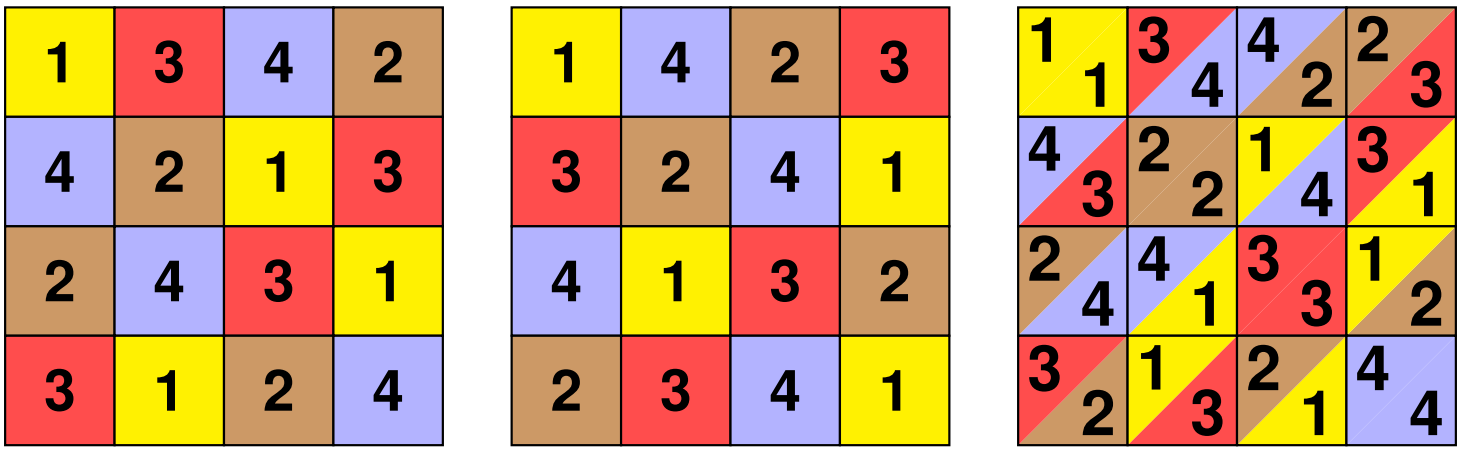
A *Latin square* of order N is a $N \times N$ matrix L such that every row and every column are permutations of $[N] = \{1, \cdots, N\}$

1	3	4	2
4	2	1	3
2	4	3	1
3	1	2	4

Orthogonal Latin Squares (OLS)

Definition

Two Latin squares L_1 and L_2 of order N are *orthogonal* if their superposition yields all the pairs $(x, y) \in [N] \times [N]$.

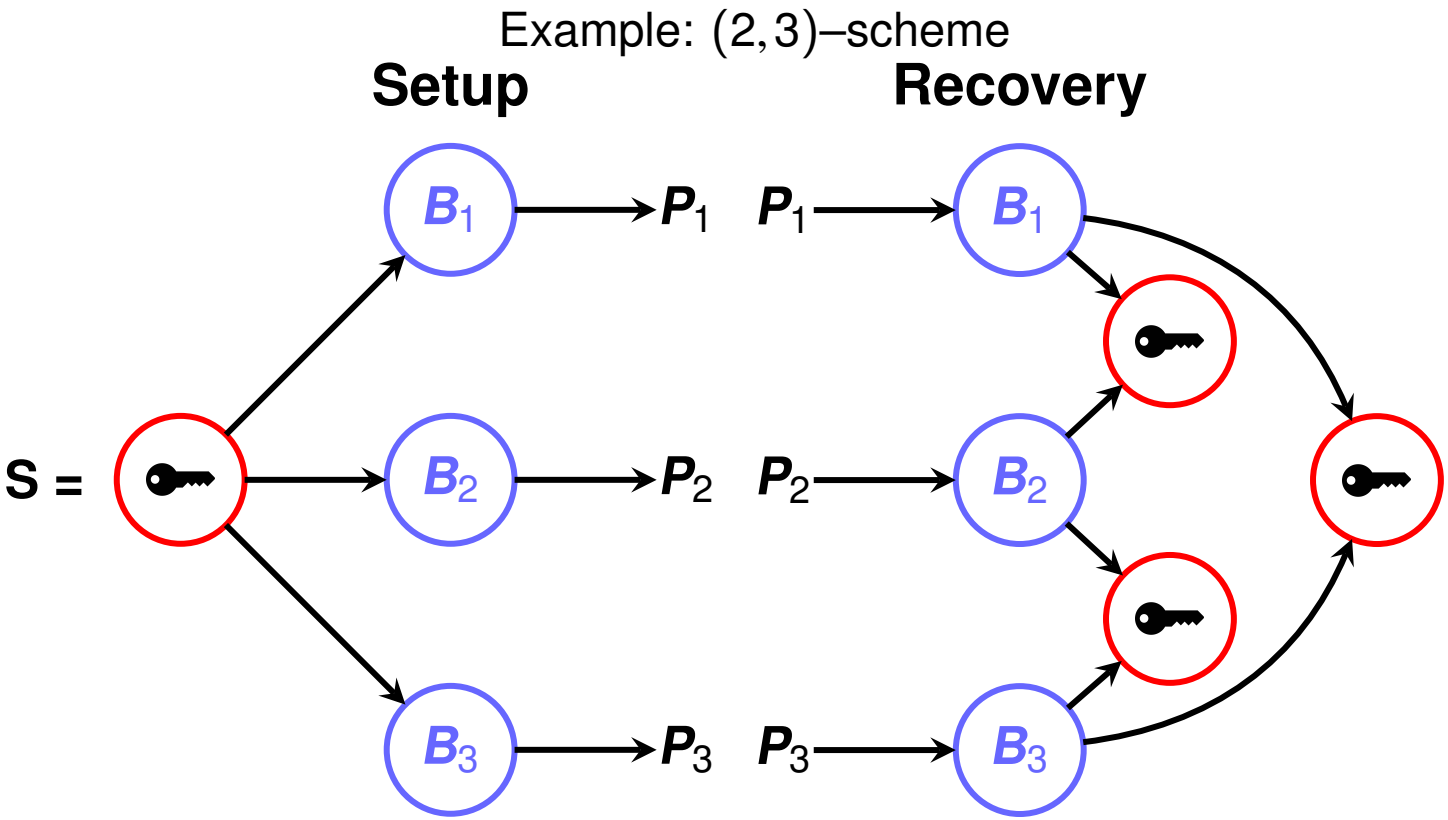


(a) L_1 (b) L_2 (c) (L_1, L_2)

n pairwise orthogonal Latin squares are denoted as n-MOLS (**Mutually Orthogonal Latin Squares**)

A Cryptographic Application of n-MOLS

(k,n) **Threshold Secret Sharing Scheme**: a **dealer** shares a **secret** S among n **players** so that at least k players out of n are required to recover S [Shamir79]



Remark: $(2,n)$–scheme $\Leftrightarrow$ set of n-MOLS

Optimization and Evolutionary Algorithms

Combinatorial Optimization

- Combinatorial Optimization Problem: map $\mathcal{P} : \mathcal{I} \to \mathcal{S}$ from a set $\mathcal{I}$ of *problem instances* to a family $\mathcal{S}$ of *solution spaces*
- $S = \mathcal{P}(I)$ is a finite set equipped with a *fitness function* $fit : S \to \mathbb{R}$, giving a score to candidate solutions $x \in S$
- Optimization goal: find $x^* \in S$ such that:

Minimization:	**Maximization:**
$x^* = argmin_{x \in S}\{fit(x)\}$	$x^* = argmax_{x \in S}\{fit(x)\}$

- Heuristic optimization algorithm: iteratively tweaks a (set of) candidate solution(s) using *fit* to drive the search

Genetic Algorithms (GA) – Genetic Programming (GP)

Optimization algorithms loosely based on evolutionary principles, introduced respectively by **J. Holland** (1975) and **J. Koza** (1989)

- Work on a **coding** of the candidate solutions
- Evolve in parallel a **population** of solutions.
- **Black-box optimization**: use only the fitness function to optimize the solutions.
- Use **Probabilistic operators** to evolve the solutions

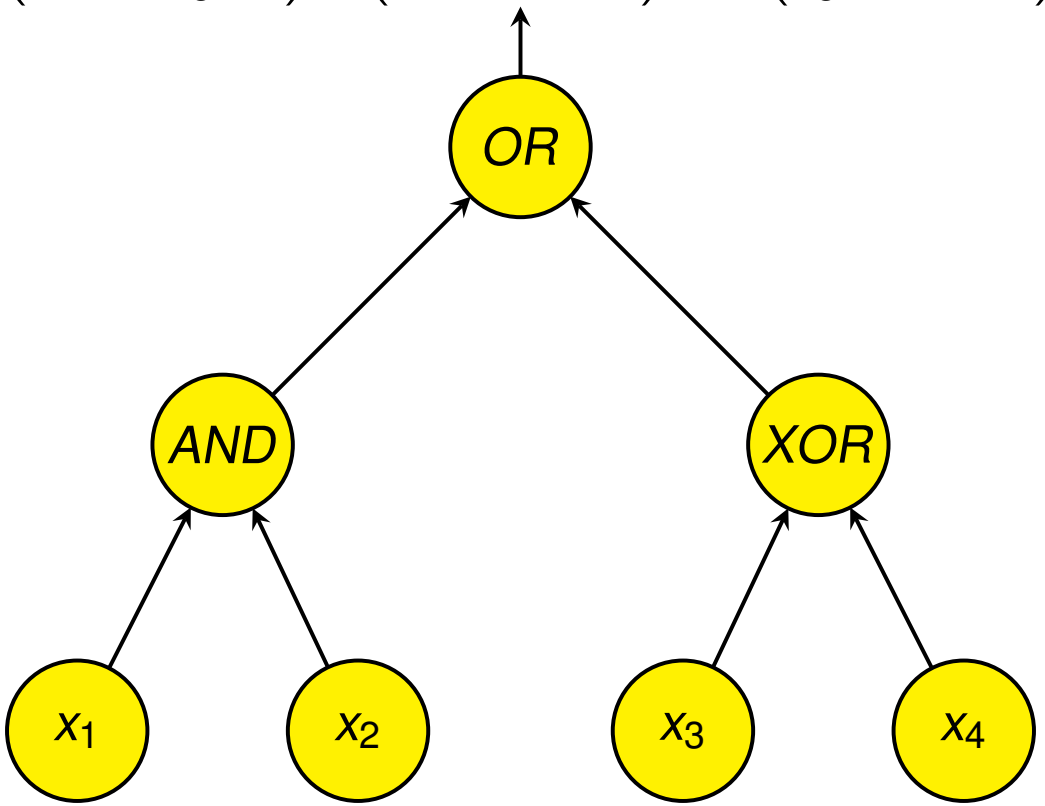
GA Encoding: Typically, an individual is represented with a fixed-length bitstring

$$\Downarrow$$

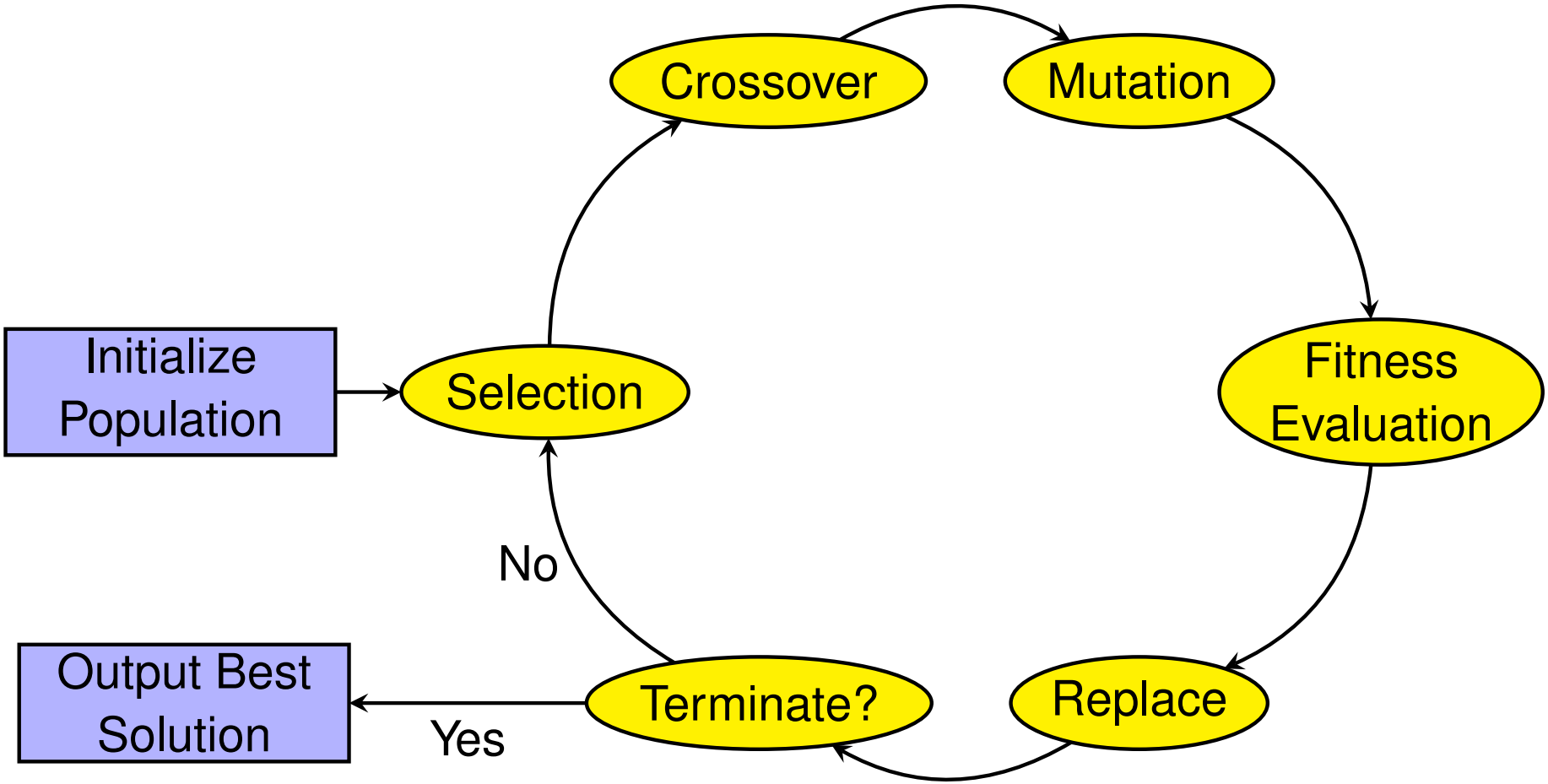
$$f(x_1, x_2, x_3) = x_1 \cdot x_2 \oplus x_1 \oplus x_2 \oplus x_3$$

- **GP Encoding**: an individual is represented by a tree
  - Terminal nodes: input variables of a program
  - Internal nodes: operators (e.g. AND, OR, NOT, XOR, ...)

$$f(x_1, x_2, x_3, x_4) = (x_1 \ AND \ x2) \ OR \ (x_3 \ XOR \ x_4)$$



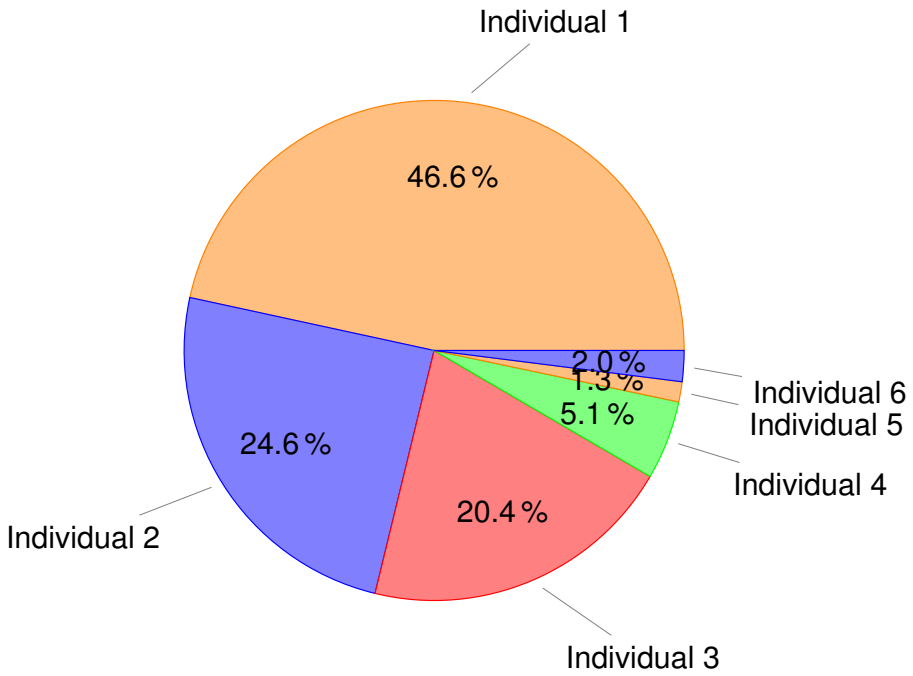
The EA Loop


Crossover
Mutation
Initialize Population
Selection
Fitness Evaluation
No
Output Best Solution
Terminate?
Replace
Yes

Selection

Roulette-Wheel Selection (RWS): the probability of selecting an individual is proportional to its fitness

Tournament Selection (TS): Randomly sample t individuals from the population and select the fittest one.



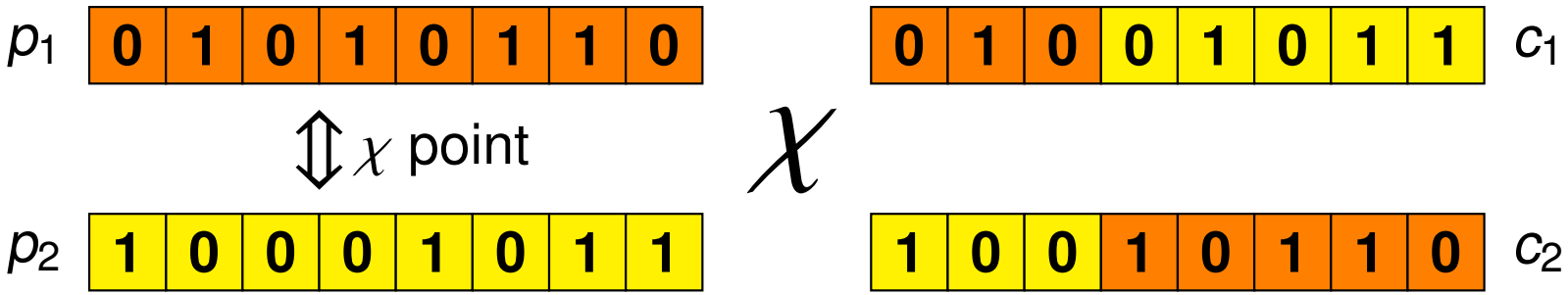
Generational Breeding: Draw as many pairs as population size

Steady-State Breeding: Select only a single pair

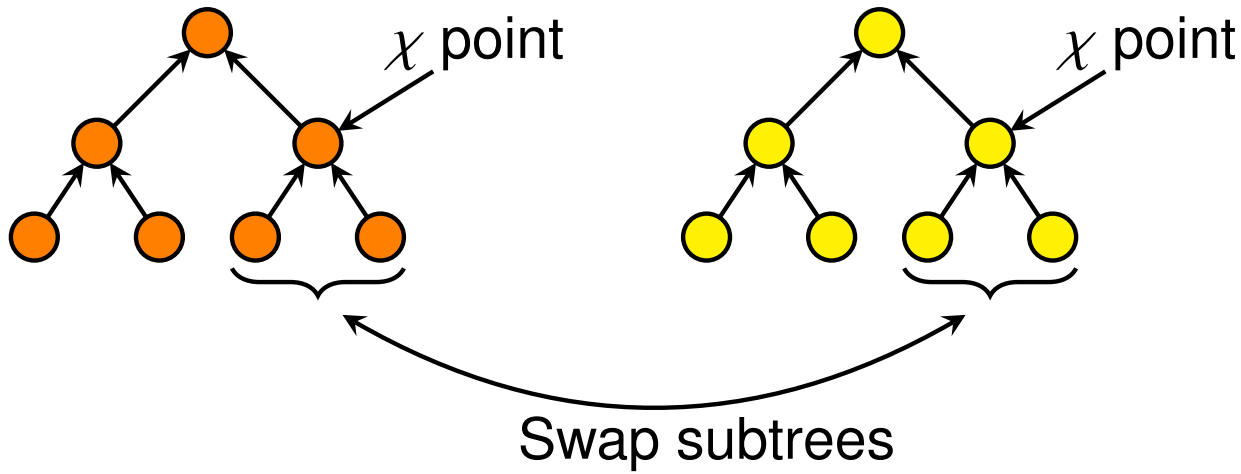
Crossover

Idea: Recombine the genes of two parents individuals to create the offspring (Exploitation)

GA Example: One-Point Crossover



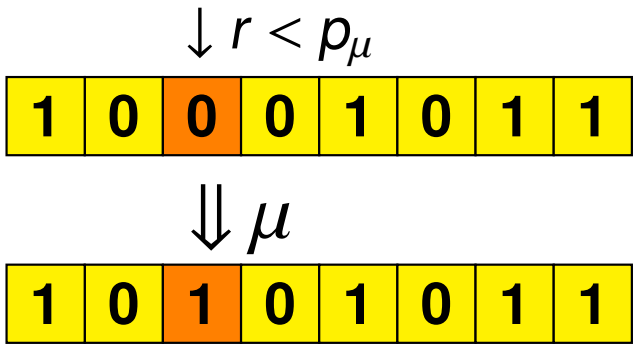
GP Example: Subtree Crossover



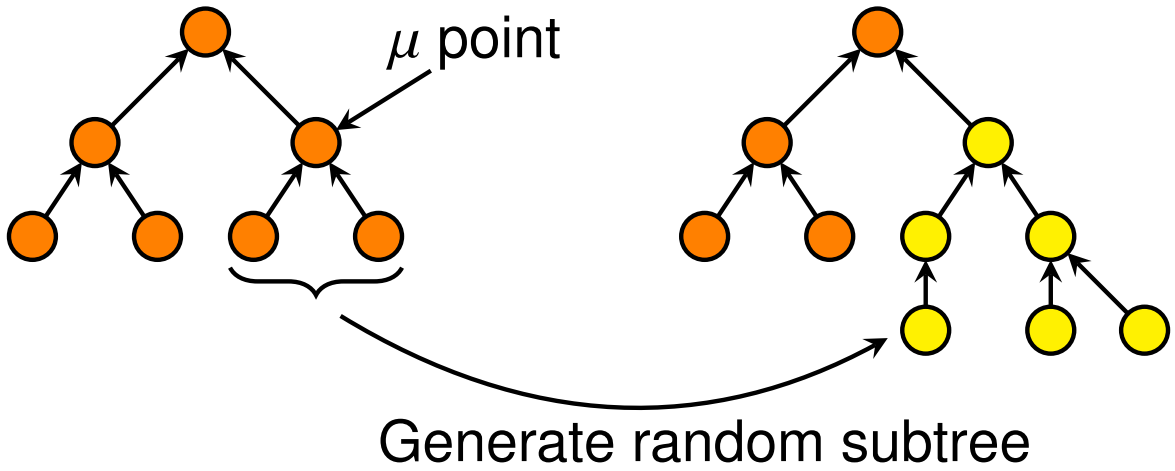
Mutation

Idea: Introduce new genetic material in the offspring (Exploration)

GA Example: Bit-flip mutation



GP Example: Subtree mutation



Replacement and Termination

- **Elitism**: keep the best individual from the previous generation
- **Termination**: several criteria such as budget of fitness evaluations, solutions diversity, ...

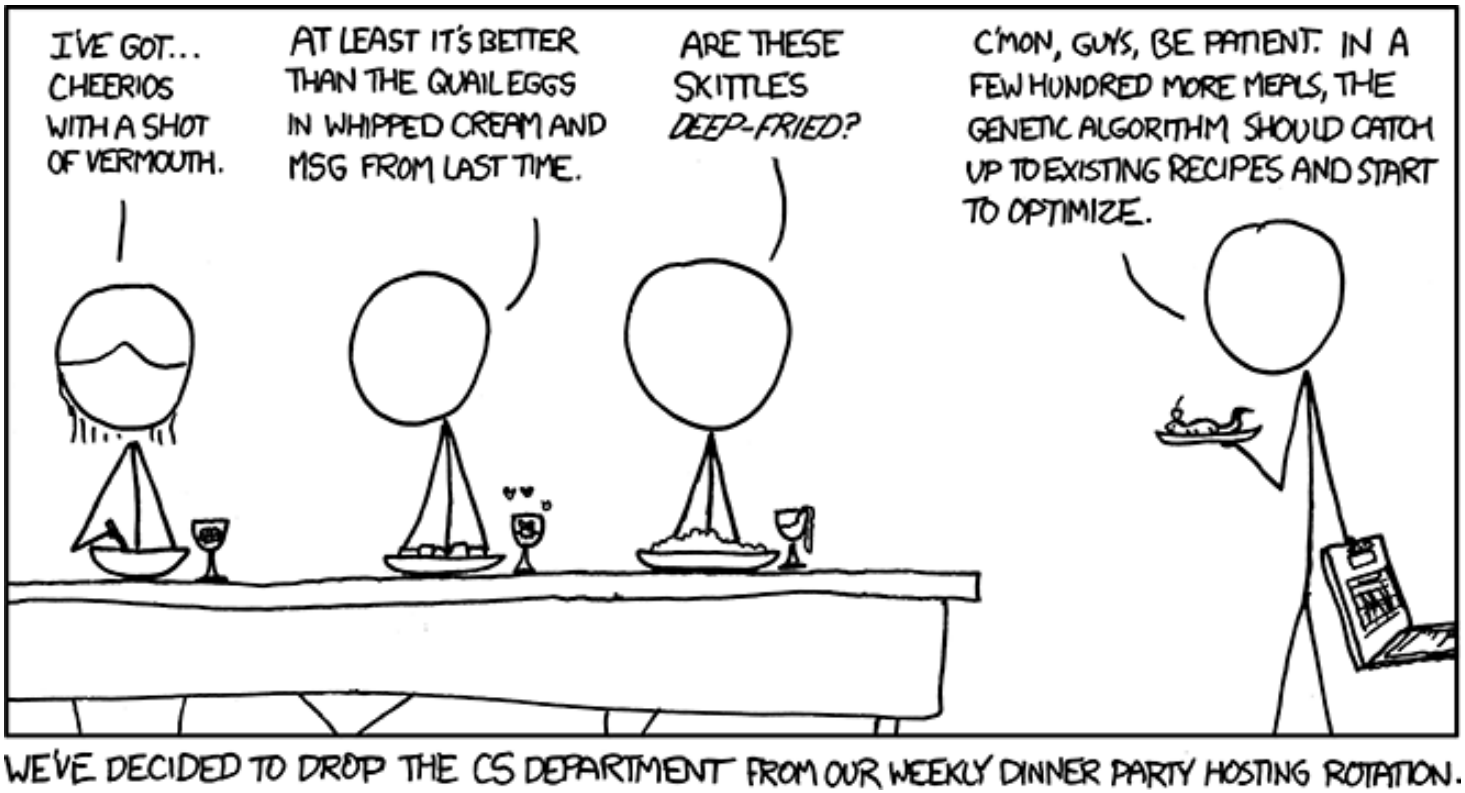


Image credit: `https://xkcd.com/720/`

Constructing OLS with EA

Construction of OLS

- Usually, performed with Algebraic methods [Stinson04]
- EA represent an interesting alternative, since they only exploit the bare definition of OLS

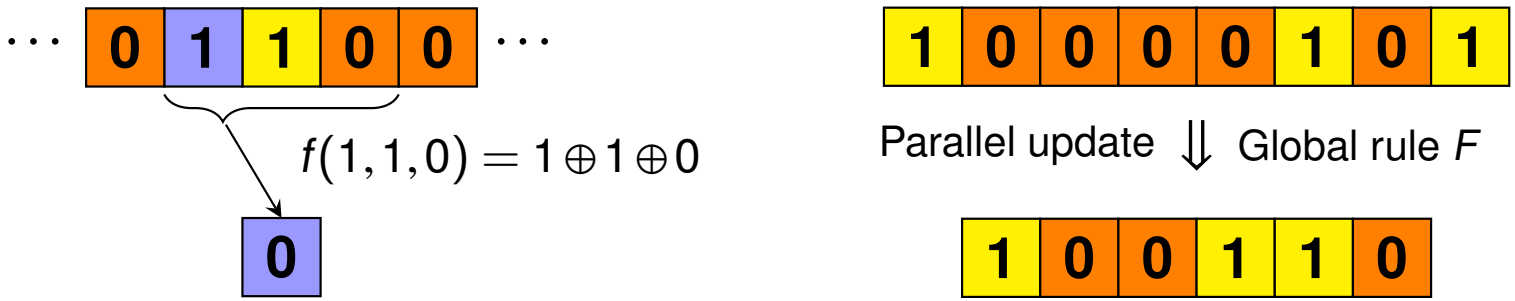
- Research questions and challenges:
  - **Representation issues**: how to encode a pair of Latin squares?
  - **Variation operators**: how to cross two individuals and still get a pair of Latin squares?
  - **Search space analysis**: the number of Latin squares/OLS is not even known for generic N

Representation: Cellular Automata (CA)

Definition

One-dimensional CA: triple $\langle m, n, f \rangle$ where $m \in \mathbb{N}$ is the number of cells on a one-dimensional array, $n \in \mathbb{N}$ is the neighborhood and $f : \{0,1\}^n \to \{0,1\}$ is the local rule.

Example: $m = 8$, $n = 3$, $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$ (Rule 150)



CA Local Rules (Boolean Functions)

- Truth table: vector Ω_f specifying $f(x)$ for all $x \in \mathbb{F}_2$

(x_1, x_2, x_3)	000	100	010	110	001	101	011	111
Ω_f	0	1	1	1	1	0	0	0

- Algebraic Normal Form (ANF): Sum (XOR) of products (AND) over the finite field $\mathbb{F}_2$

$$f(x_1, x_2, x_3) = x_1 \cdot x_2 \oplus x_1 \oplus x_2 \oplus x_3$$

- Affine function: $l(x_1, \cdots, x_n) = a \oplus a_1 x_1 \oplus \cdots \oplus a_n x_n$, $a, a_i \in \{0, 1\}$
- Nonlinearity of f: Hamming distance of $\Omega(f)$ from the set of all affine functions

Latin Squares through Bipermutive CA (1/2)

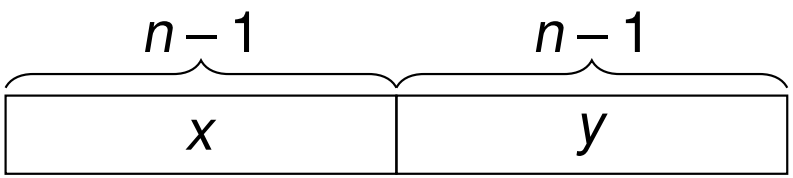
- Idea: determine which CA induce orthogonal Latin squares
- Bipermutive CA: local rule f is defined as

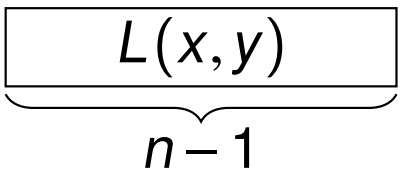
$$f(x_1, \cdots, x_n) = x_1 \oplus \varphi(x_2, \cdots, x_{n-1}) \oplus x_n$$

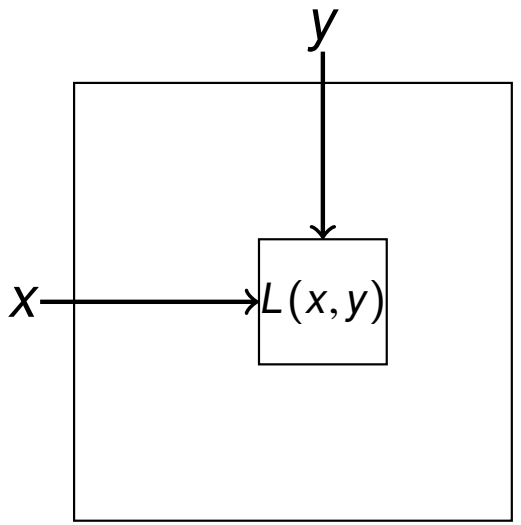
- $\varphi : \{0,1\}^{n-2} \to \{0,1\}$: generating function of f

Lemma ([Mariot19])

Let $\langle 2(n-1), n, f \rangle$ be a CA with bipermutive rule. Then, the global rule F generates a Latin square of order $N = 2^{n-1}$

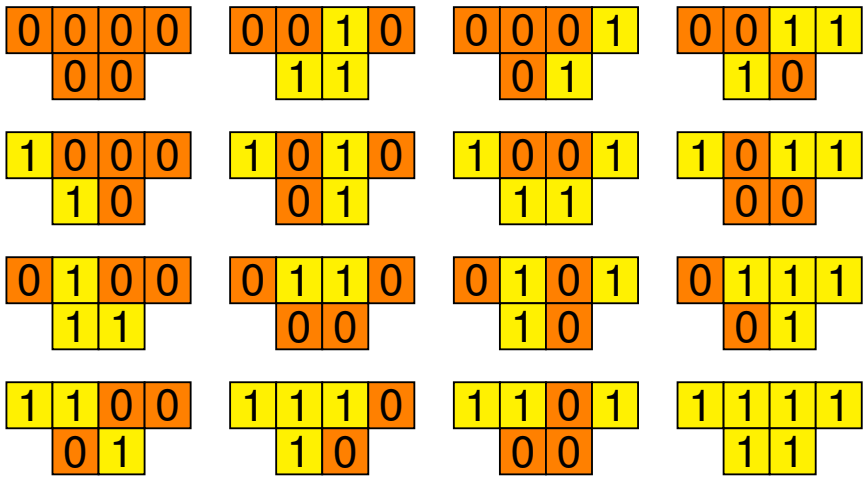






Latin Squares through Bipermutive CA (2/2)

- Example: CA $\langle 4, 1, f \rangle$, $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$ (Rule 150)
- Encoding: $00 \mapsto 1, 10 \mapsto 2, 01 \mapsto 3, 11 \mapsto 4$



(a) Rule 150 on 4 bits

1	4	3	2
2	3	4	1
4	1	2	3
3	2	1	4

(b) Latin square L_{150}

Goals and Motivations

- Construction of OLS solved for linear CA [Mariot19]

Goal: Design OLS based on CA by evolving pairs of nonlinear bipermutive local rules through GA and GP

Three motivations:

- **Theoretical:** Understand the mathematical structure of the space of nonlinear CA-based OLS
- **Applications:** Design of cheater-immune SSS [Tompa88]
- **EC perspective:** Source of new problems for EA

GA Encoding: Single Bitstring

- Let $\Omega(\varphi), \Omega(\gamma)$ be a pair of generating functions truth tables, and let $||$ denote concatenation

$$\textbf{First GA encoding}: enc_1(\varphi, \gamma) = \Omega(\varphi)||\Omega(\gamma)$$

Example:

$$\varphi(x_1, x_2, x_3) = x_1 \oplus x_3 \Rightarrow \Omega(\varphi) = (0,1,0,1,1,0,1,0)$$
$$\gamma(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3 \Rightarrow \Omega(g) = (0,1,1,0,1,0,0,1)$$
$$enc_1(\varphi, \gamma) = (0,1,0,1,1,0,1,0,0,1,1,0,1,0,0,1)$$

- Classic GA variation operators like one-point crossover and bit-flip mutation are applied in this case

GA & GP Encodings: Double Bitstring/Double Tree

- Idea: Keep the generating functions separated and evolve them independently

  Second GA encoding: $enc_2(\varphi, \gamma) = (\Omega(\varphi), \Omega(\gamma))$

- We use the same idea for GP: the genotype is composed of two trees $T(\varphi)$ and $T(\gamma)$ representing φ and γ

  GP encoding: $enc_{GP}(\varphi, \gamma) = (T(\varphi), T(\gamma))$

- Classic GA and GP variations operators are applied independently on each of the two components

GA Encoding: Balanced Quaternary Strings (1/2)

Definition

$f, g : \{0,1\}^n \to \{0,1\}$ are pairwise balanced (PWB) if

$$\left|(f,g)^{-1}(0,0)\right| = \left|(f,g)^{-1}(1,0)\right| = \\ = \left|(f,g)^{-1}(0,1)\right| = \left|(f,g)^{-1}(1,1)\right| = 2^{n-2}$$

Example:

- $f(x_1, x_2, x_3) = x_1 \oplus x_3$ (Rule 90)
- $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$ (Rule 150)

$$\Omega(f) = (0,1,0,1,1,0,1,0) \ ,$$
$$\Omega(g) = (0,1,1,0,1,0,0,1) \ .$$

Each of the pairs $(0,0),(1,0),(0,1),(1,1)$ occurs $2^{3-2} = 2$ times

GA Encoding: Balanced Quaternary Strings (2/2)

- Theoretical results on PWB functions [Mariot17b]:
  - Two bipermutive CA generate OLS $\Rightarrow$ the local rules are PWB
  - Generating functions are PWB $\Rightarrow$ the local rules are PWB

Third GA encoding: $enc_3(\varphi, \gamma)$ is a quaternary string of length 2^{n-2} where each number from 1 to 4 occurs 2^{n-4} times

Example: $n = 5, (0,0) \mapsto 1, (1,0) \mapsto 2, (0,1) \mapsto 3, (1,1) \mapsto 4$

$$\Omega(\varphi) = (0,1,0,1,1,0,1,0)$$
$$\Omega(\gamma) = (0,1,1,0,1,0,0,1)$$
$$enc_3(\varphi, \gamma) = (1,4,3,2,4,1,2,3)$$

- Balancedness-preserving variation operators for GA:
  - Crossover: use counters to keep track of the multiplicities of the 4 values in the offspring
  - Mutation: use a swap-based operator

Fitness Functions (1/2)

- $\#rep(L_1, L_2)$: Number of repetitions of each pair occurring in the superposition of Latin squares L_1 and L_2

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

(a) L_1

4	3	2	1
3	2	1	4
2	1	4	3
1	4	3	2

(b) L_2

1,4	2,3	3,2	1,4
2,3	3,2	4,1	1,4
3,2	4,1	1,4	2,3
4,1	1,4	2,3	3,2

(c) $\#rep(L_1, L_2) = 12$

- Let φ, γ be the generating functions of two bipermutive CA, and let L_φ, L_γ be the associated Latin squares

First fitness function: minimize $fit_1(\varphi, \gamma) = \#rep(L_\varphi, L_\gamma)$

Fitness Functions (2/2)

- **Remark**: fit_1 does not consider the nonlinearity of φ and γ!
- Nonlinearity penalty factor:

$$NlPen(\varphi,\gamma) = \begin{cases} 0 \ , & \text{if } Nl(\varphi) > 0 \ \ AND \ \ Nl(\gamma) > 0 \\ 1 \ , & \text{if } Nl(\varphi) = 0 \ \ XOR \ \ Nl(\gamma) = 0 \\ 2 \ , & \text{if } Nl(\varphi) = 0 \ \ AND \ \ Nl(\gamma) = 0 \end{cases}$$

Second fitness function: minimize

$$fit_2(\varphi,\gamma) = \#rep(L_\varphi, L_\gamma) + NlPen(\varphi,\gamma) \cdot N^2$$

- The N^2 scaling factor balances the range of $\#rep(L_\varphi, L_\gamma)$

Experimental settings

Common Parameters:

- Problem instances: rules of $n = 7$ and $n = 8$ variables
- Termination condition: 300000 fitness evaluations
- Each experiment is repeated over 50 independent runs
- Selection operator: steady-state with 3-tournament operator

GA Parameters:

- Population size: 30 individuals
- Crossover and mutation probabilities: $p_c = 0.95$, $p_m = 0.2$

GP Parameters:

- Boolean operators: AND, OR, XOR, XNOR, NOT, IF
- Population size: 500 individuals
- Mutation probability: $p_m = 0.5$

Results

- **Main finding**: GP always converges to an optimal solution, but only to linear ones with fit_1

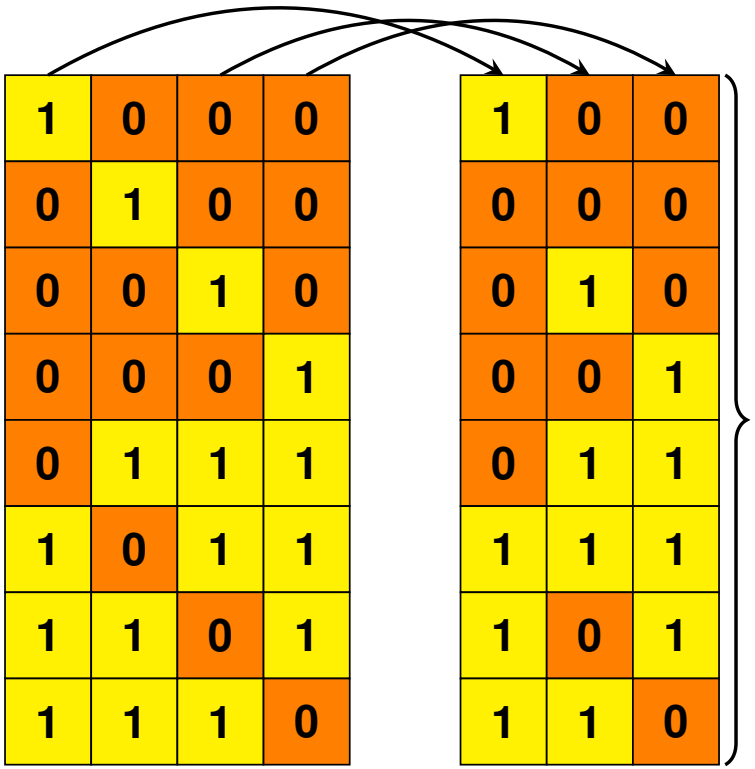
Exp.	avg fit	std fit	#opt	#lin	#nlin
$(GA, 7, enc_1)$	520.32	360.16	12/50	0	12
$(GA, 7, enc_2)$	565.44	389.03	15/50	0	15
$(GA, 7, enc_3)$	392.64	328.47	18/50	0	18
$(GA, 8, enc_1)$	4165.44	604	1/50	0	1
$(GA, 8, enc_2)$	4222.16	125.03	0/50	0	0
$(GA, 8, enc_3)$	4696.48	135.51	0/50	0	0
$(GP, 7, fit_1)$	0	0	50/50	50	0
$(GP, 7, fit_2)$	0	0	50/50	0	50
$(GP, 8, fit_1)$	0	0	50/50	47	3
$(GP, 8, fit_2)$	0	0	50/50	0	50

- Results published in GECCO 2017 [Mariot17a]

Constructing OA with EA

Orthogonal Arrays (OA)

- (N,k,t,λ) **Orthogonal Array**: $N \times k$ binary matrix A such that each binary t-uple occurs λ times in each $N \times t$ submatrix.



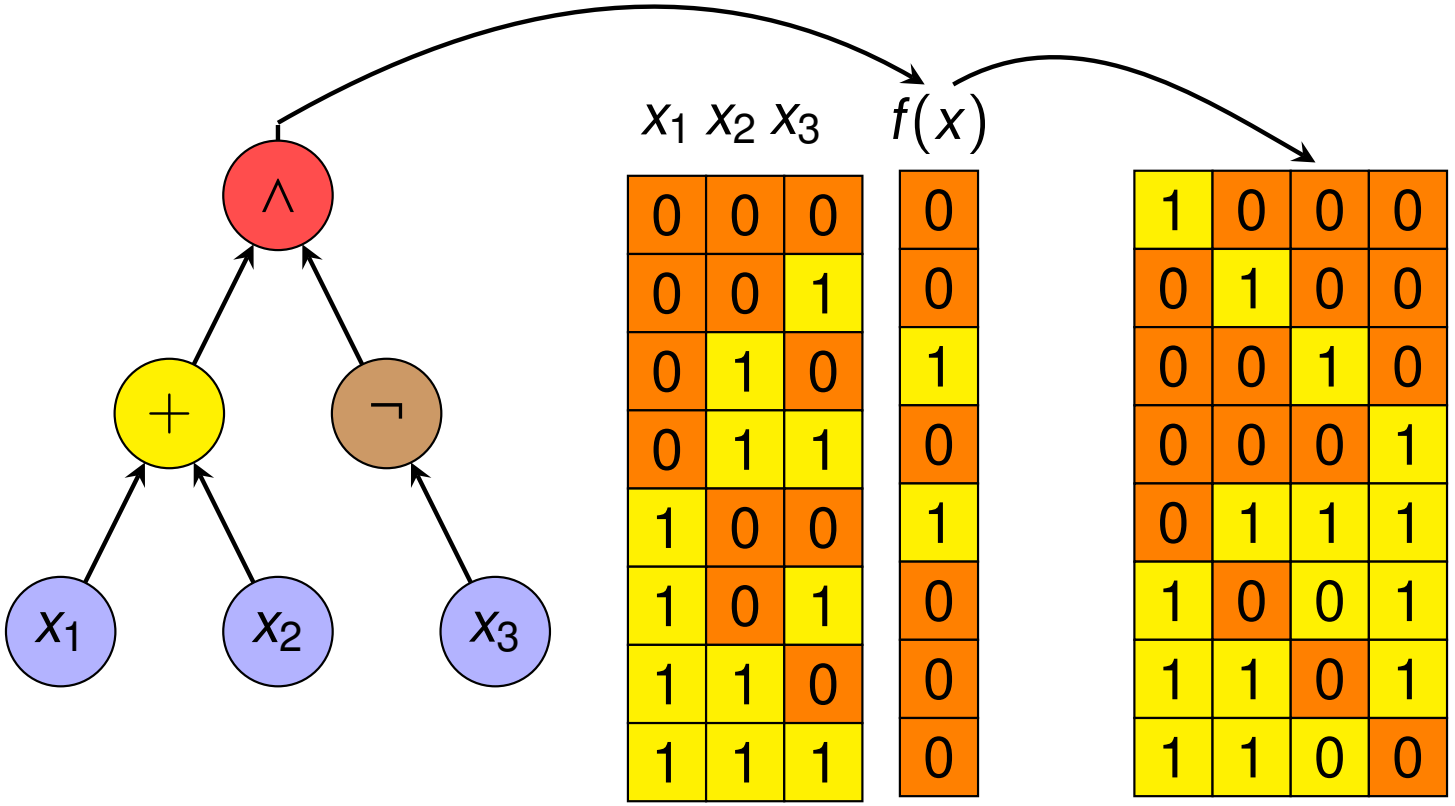
Example: OA $(8,4,3,1)$

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

- Applications: designs of experiments, error-correcting codes
- **Goal**: Construct OA with Evolutionary Algorithms (EA)

Solutions Encoding

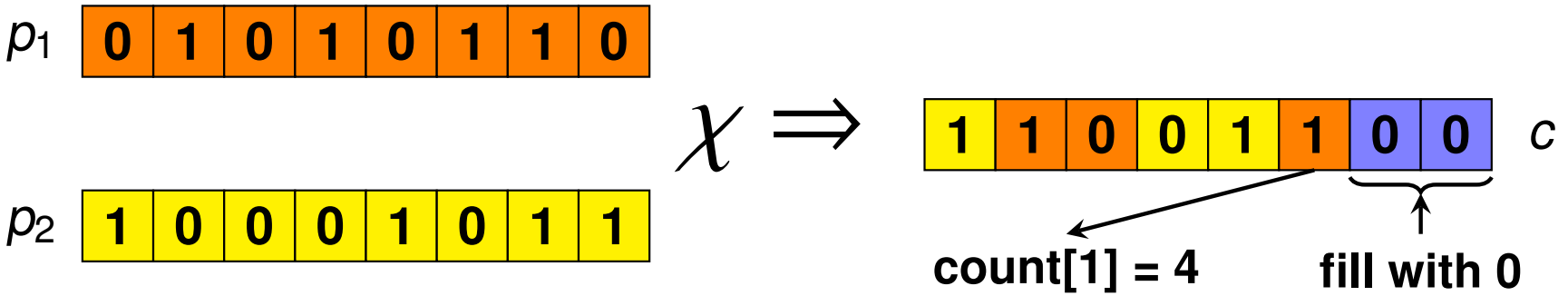
- Each column is the truth table of a n-variable Boolean function
- For GP, the truth table is synthesized from the tree of the individual



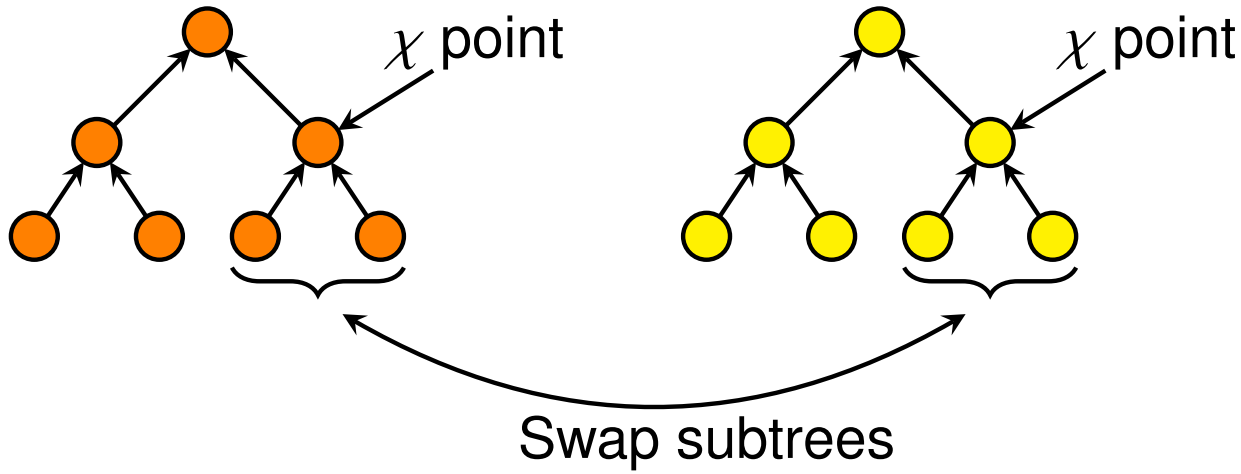
- Crossover and mutation are applied **column-wise**

Crossover Operators

- **Remark**: Each column of an OA must be balanced
- **GA Crossover Idea:** Use *counters* to keep track of the multiplicities of zeros and ones [Millan98]

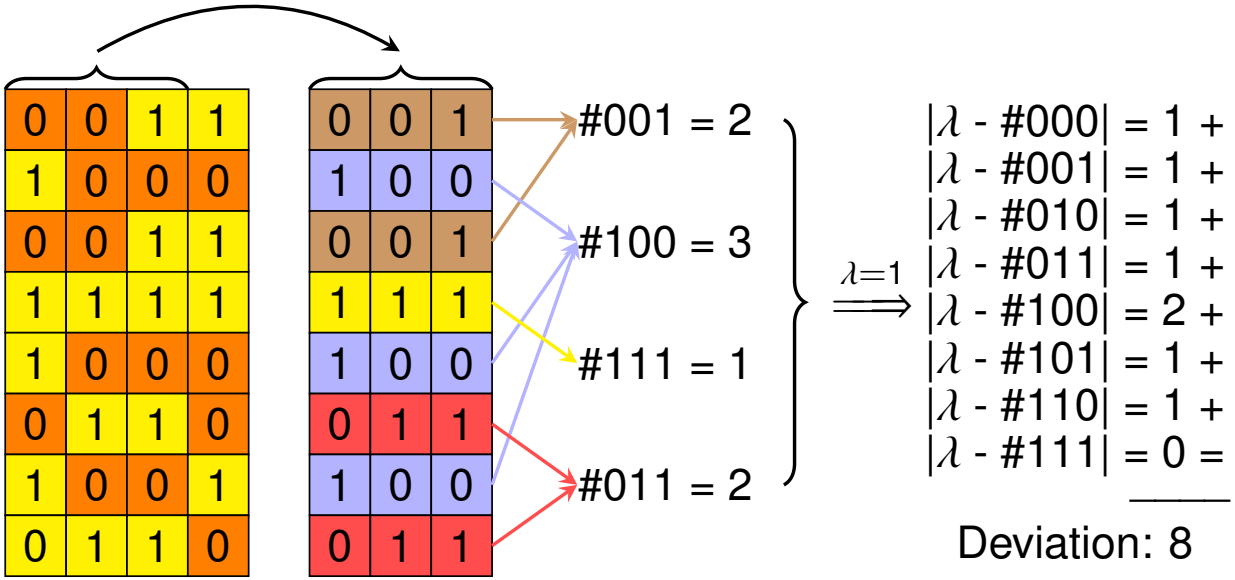


- **For GP**: Use standard subtree crossover



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 Setting

- Problem instances considered:

	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, 30 independent runs for each instance

Results

- **Main finding:** GP outperforms by far GA wrt success rate

	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

- Results published in PPSN 2018 [Mariot18]

Conclusions and Future Directions

Conclusions

Construction of OLS:

- GP seems a better heuristic than GA to construct OLS and OA, although it explores a larger solution space
- ... but under fit_1 on the OLS problem GP always converges to linear solutions!
- GA has worse performances, but always finds nonlinear solutions

Construction of OA:

- GP still outperforms GA, but does not converge on all considered instances
- Is GP learning a "linear" structure of solutions, as with OLS?

Future Directions

Open problems:

- Understand the difference in performances between GA and GP (e.g. via fitness landscape analysis)
- Investigate the preference of GP for linear solutions on OLS
- Consider other approaches to OLS and OA design with EA (e.g., incremental construction)
- Apply EA to construct other kind of CD (e.g. disjunct matrices, permutation codes, ...)

References

- [Mariot19] Mariot, L., Gadouleau, M., Formenti, E., Leporati, A.: Mutually orthogonal latin squares based on cellular automata. Des. Codes Cryptogr. (2019) doi:10.1007/s10623-019-00689-8
- [Mariot18] Mariot, L., Picek, S., Jakobovic, D., Leporati, A.: Evolutionary Search of Binary Orthogonal Arrays. In: Auger, A., Fonseca, C.M., Lourenço, N., Machado, P., Paquete, L., Whitley, D. (eds.): PPSN 2018 (I). LNCS vol. 11101, pp. 121–133. Springer (2018)
- [Mariot17a] Mariot, L., Picek, S., Jakobovic, D., Leporati, A.: Evolutionary Algorithms for the Design of Orthogonal Latin Squares based on Cellular Automata. In: Proceedings of GECCO'17, pp. 306–313 (2017)
- [Mariot17b] Mariot, L., Formenti, E., Leporati, A.: Enumerating Orthogonal Latin Squares Generated by Bipermutive Cellular Automata. In: Dennunzio, A., Formenti, E., Manzoni, L., Porreca, A. E. (eds.): AUTOMATA 2017. LNCS vol. 10248, pp. 151–164. Springer (2017)
- [Millan98] Millan, W., Clark, J., Dawson, E.: Heuristic Design of Cryptographically Strong Balanced Boolean Functions. EUROCRYPT 1998: pp. 489–499 (1998)
- [Shamir79] Shamir, A.: How to share a secret. Commun. ACM 22(11):612–613 (1979)
- [Stinson04] Stinson, D. R.: Combinatorial designs. Springer (2004)
- [Tompa88] Tompa, M., Woll, H.: How to share a secret with cheaters. J. Cryptology 1(2), 133–138 (1988)