On the Importance of Initial Solutions Selection in Fault Injection

FDTC 2021

Marina Krček, Daniele Fronte, Stjepan Picek

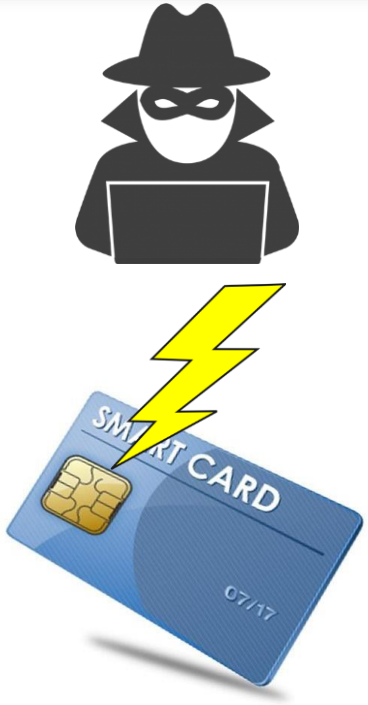
Fault Injection (FI) attacks

- Introducing faults (voltage, laser, etc.)
- Analyzing the device responses to get the secret information

Differential Fault Analysis (DFA)

- Analyzing correct/faulty output pairs and comparing them

Security evaluation of hardware devices



Focus of this work

- Efficiently injecting faults

Introducing faults

Voltage glitching

Clock glitching

Electromagnetic fault injection (EMFI)

Laser fault injection (LFI)

Problem: choosing multiple parameters required for effective faults

Parameter search space size

EMFI*, 5 parameters: 24 × 24mm chip size with a 0.05mm repositioning error

- at ≈ 0.16 sec per point, an exhaustive search would take 29 203 years to conduct

LFI, 5 parameters: reduced intervals

- at ≈ 0.16 sec per point, for 31,7 million possibilities it would take ≈ 59 days

* Maldini, A., Samwel, N., Picek, S., Batina, L.: Genetic algorithm-based electromagnetic fault injection. In: 2018 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). pp. 35-42. IEEE (2018)

Current search algorithms

Exhaustive search, random search

- In areas known to be sensitive to the fault injection (based on previous experience)

Issues:

- Time-consuming and non-deterministic process
- Measurement results obtained on one setup cannot easily be reproduced by another
- Expertise not easy to obtain

Focus of this work

- Efficient exploring parameter space

Laser Fault Injection (LFI) attacks

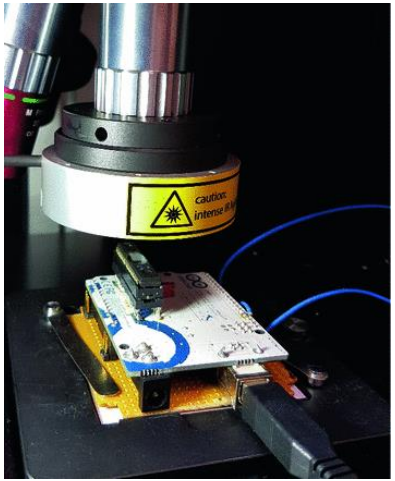
Parameters

Location on the device: x, y

Distance from the lens of the microscope

Spot size

Laser settings: wavelength, pulse width, repetition rate, duration, power, etc.



Parameter selection using Evolutionary Algorithms

Already done for voltage glitching and EMFI

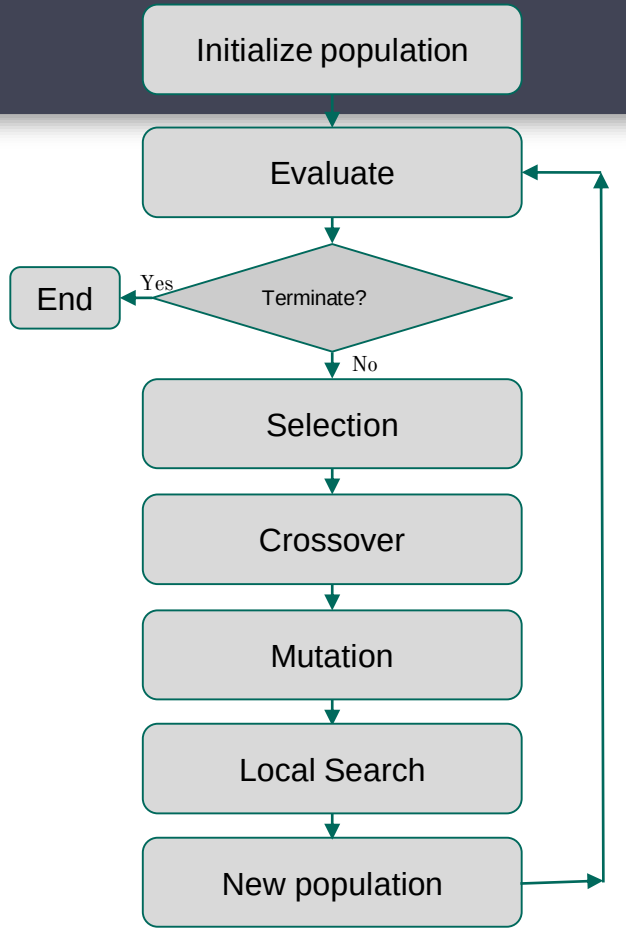
Using Genetic Algorithms (GA) and Memetic Algorithms (MA)

- Population-based algorithms
- Evolution through iterations
- MA: GA + local search

How to use Memetic Algorithm for LFI parameter selection?

Memetic Algorithm (MA) for LFI

Simple genetic algorithm with Hooke-Jeeves algorithm for local search



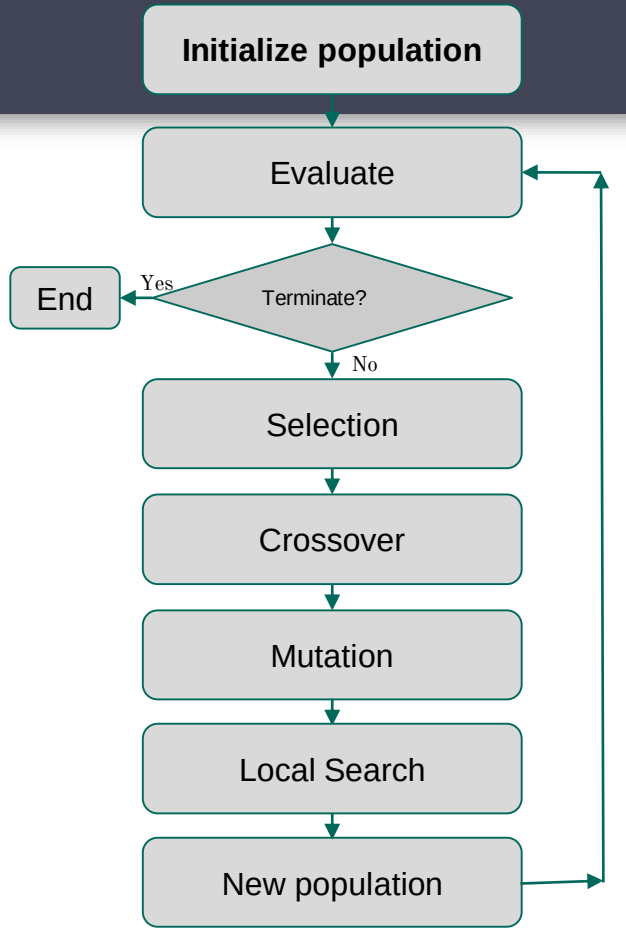
Initial population

Solutions in our case

Set of parameters (x, y, delay, laser pulse width, and laser intensity)

Population of size n

x_1, y_1, d_1, pw_1, i_1	f_1
x_2, y_2, d_2, pw_2, i_2	f_2
...	...
x_n, y_n, d_n, pw_n, i_n	f_n



Evaluation function

Conduct the laser shot several times for all solutions

Include sorting algorithm:

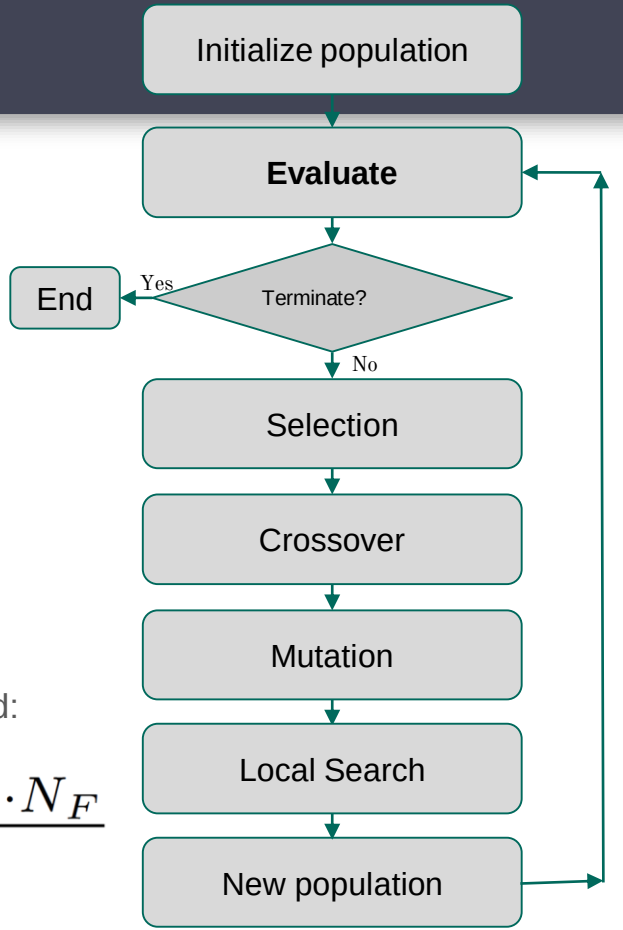
- Sort the x and y coordinates
- **greedy algorithm** based on **Manhattan distance** between the points

Device response categorized into one of the fault classes: *pass*, *mute*, *fail* and *changing*

Fault class	Fitness value
PASS	2
MUTE	5
FAIL	10

Fitness for *changing* is calculated:

$$\frac{f_P \cdot N_P + f_M \cdot N_M + f_F \cdot N_F}{N_P + N_M + N_F}$$

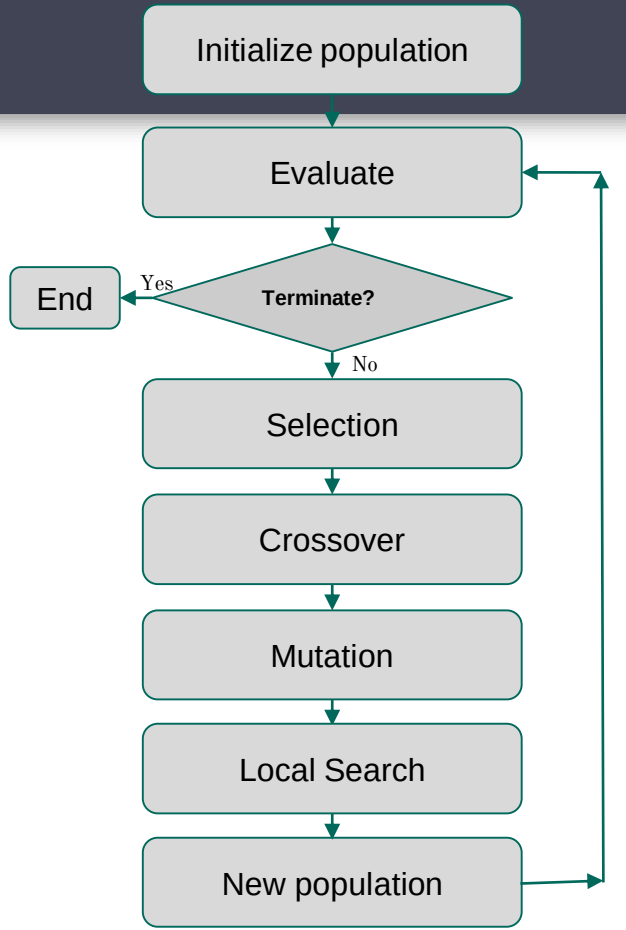


Termination condition

We consider only the number of iterations

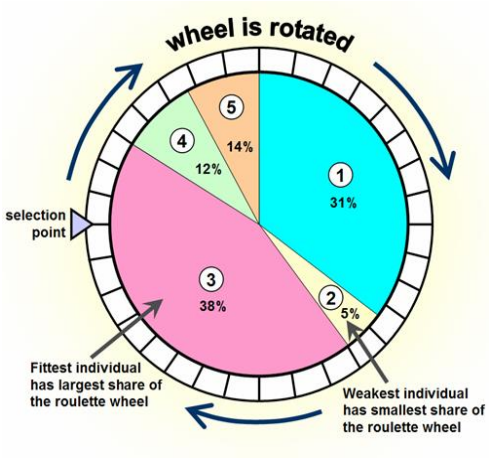
- Set by user
- When maximum number of iteration is reached, the algorithm ends

Iteration: GA operators + Local search

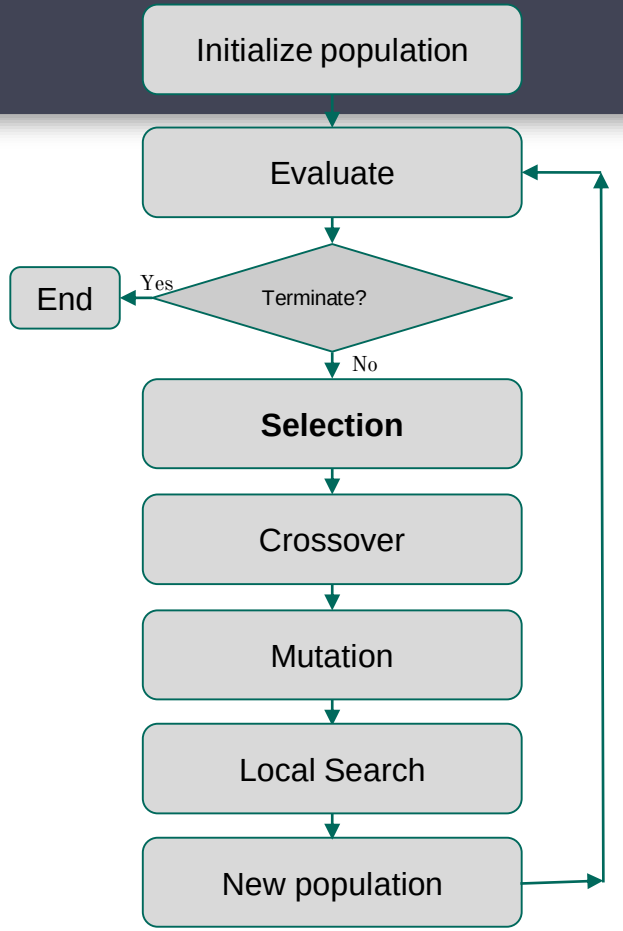


Selection operator

Fitness proportionate selection (roulette wheel)



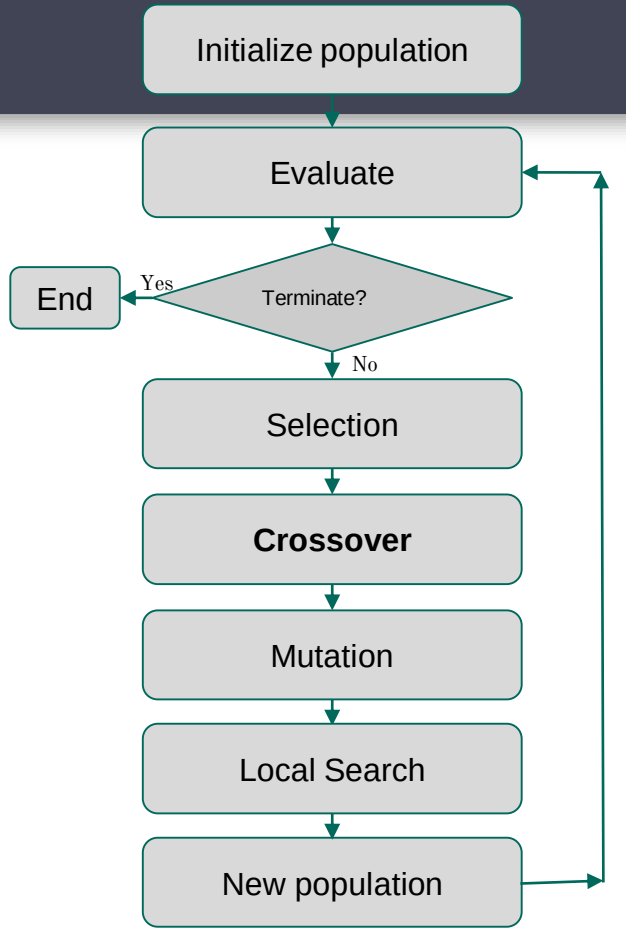
$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$



Crossover operator

Average crossover

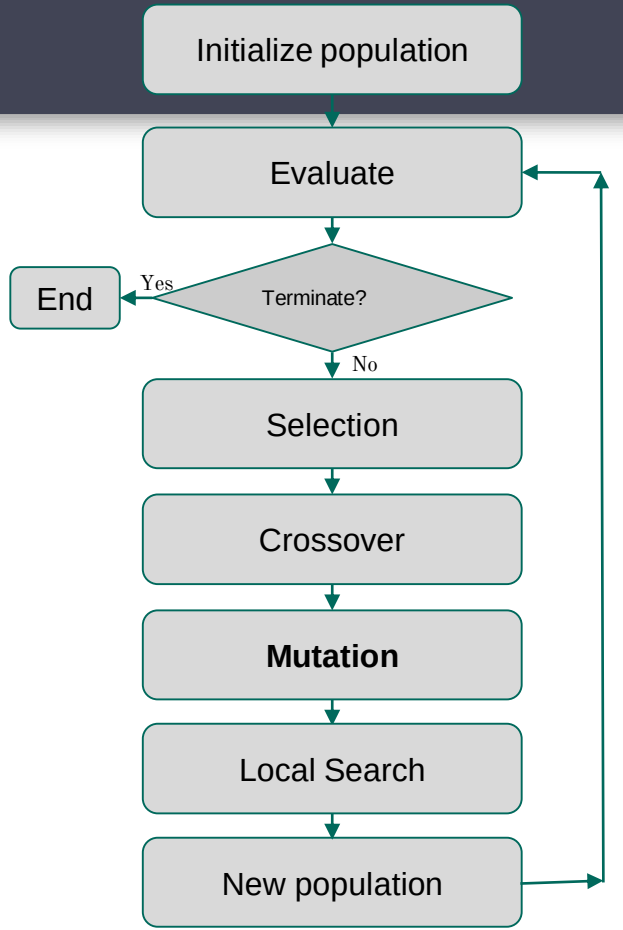
Parent 1	5	3	3	2	3
Parent 2	5	4	7	6	5
Offspring	5	3	5	4	4



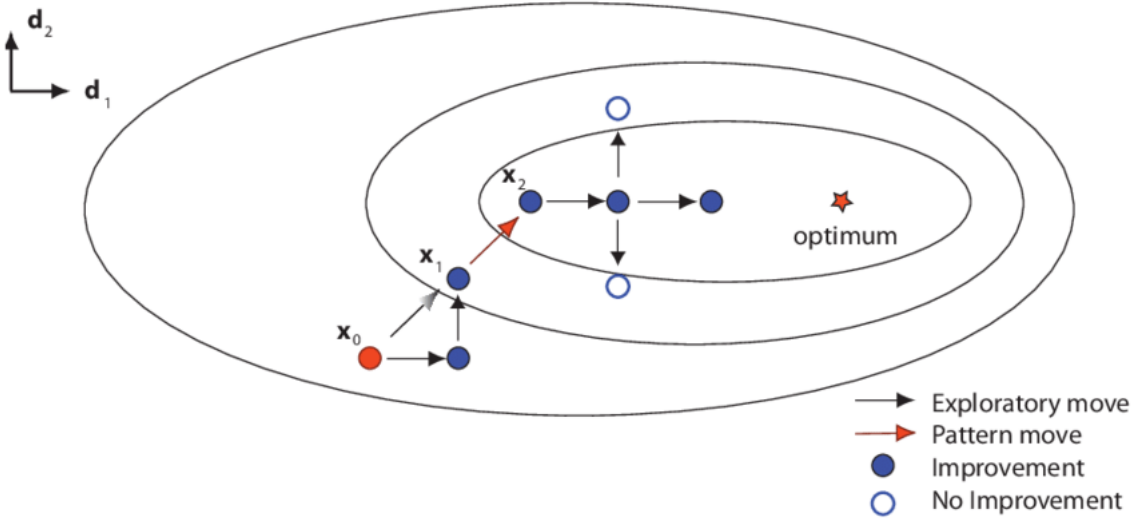
Mutation

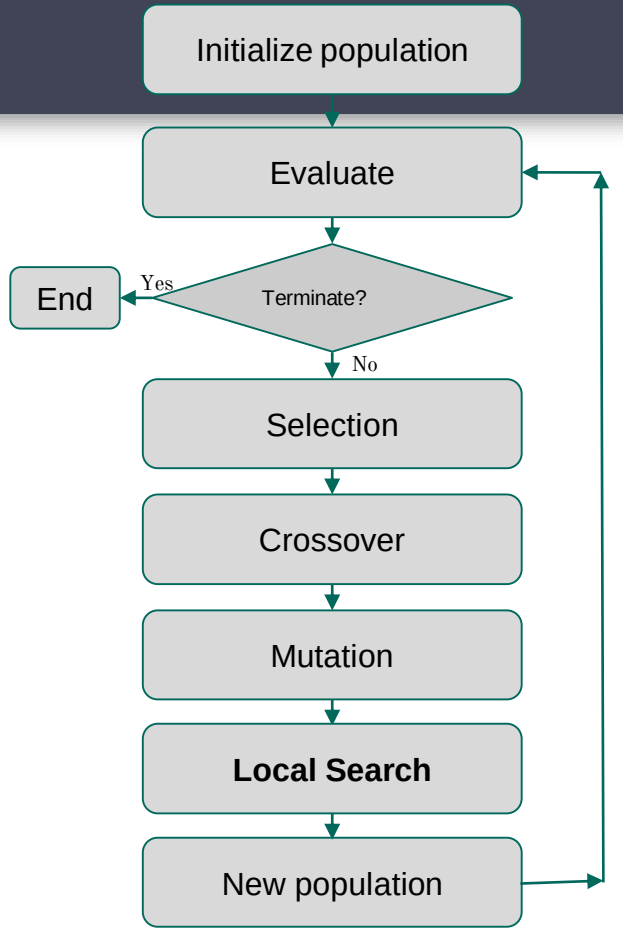
Uniform mutation

5	3	5	4	4
5	2	5	4	4



Local Search: Hooke-Jeeves algorithm



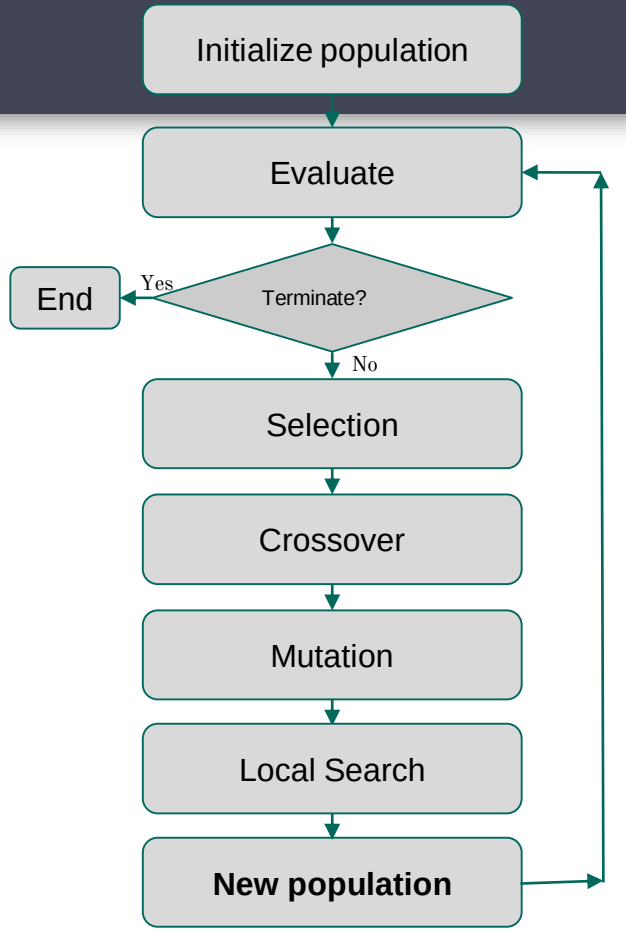


Solutions with fitness > 85% of the maximum fitness (10)
At most 3 are randomly selected for local search

Elitism

Best solutions taken to a new generation without modification

- Elite size



Experimental Setup

Test chip from STMicroelectronics using 40nm technology

The laser faults are injected while loading a data word into a register from non-volatile memory (NVM)

Parameters: subsets of the available values

Number of measurements per solution: 5

Number of runs for each test case: 5

```
...
trigger_event()
load_register() // injection here
read_register()
...
```

Comparing Random Search and MA

Random Search: no duplicate solutions

Memetic algorithm with random initialization: number of iterations 200, population size 36, elite size 2, mutation probability 0.05

Algorithm	**Tested parameters**	**Fails (%)**	**Changing (%)**	**Mute (%)**	**Pass (%)**
Random	9130	0.9	2.58	2.54	93.98
Memetic	9034.6	**31.03**	7.11	3.64	58.22

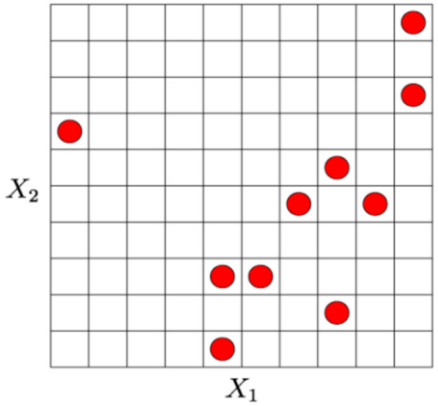
Initialization methods

Random initialization - most often

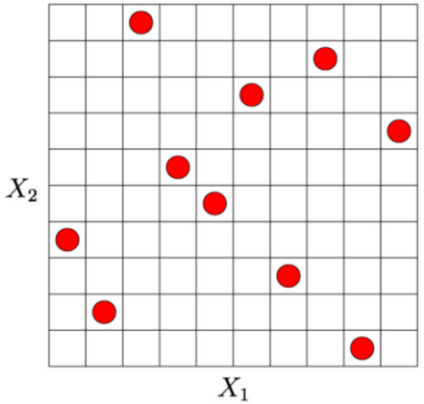
Latin Hypercube Sampling (LHS)

LHS with multidimensional uniformity (LHSMDU)

Taguchi method



Monte Carlo Simulation



Latin Hypercube Sampling

1	1	1
2	2	1
1	2	2
2	1	2

Example of OA with strength 2, number of experiments 4, 3 factors, and 2 factor levels.

https://pypi.org/project/pyDOE2/
https://pypi.org/project/lhsmdu/
https://pypi.org/project/OApackage/

Comparison with smaller population

Population size 36, elite size 2, mutation probability 0.05, number of iterations 200

Taguchi method, OA of 36 samples, with strength 2, and factor levels 3,3,3,2,2

Initialization	Tested parameters	Fails (%)	Changing (%)	Mute (%)	Pass (%)
Random	9034.6	31.03	7.11	3.64	58.22
LHS	8812.8	28.8	7.73	4.59	58.88
LHSMDU	**9605**	**33.92**	7.67	3.02	55.4
Taguchi	9070.2	31.58	6.87	3.33	58.23

Comparison with bigger population

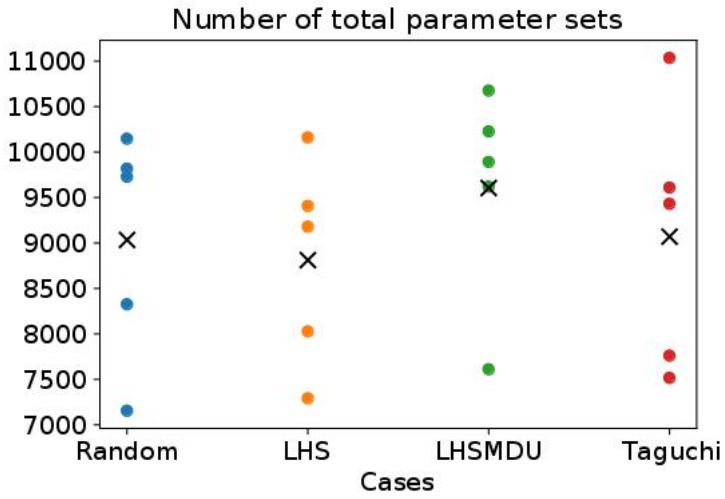
Population size 128, elite size 10, mutation probability 0.05, number of iterations 200

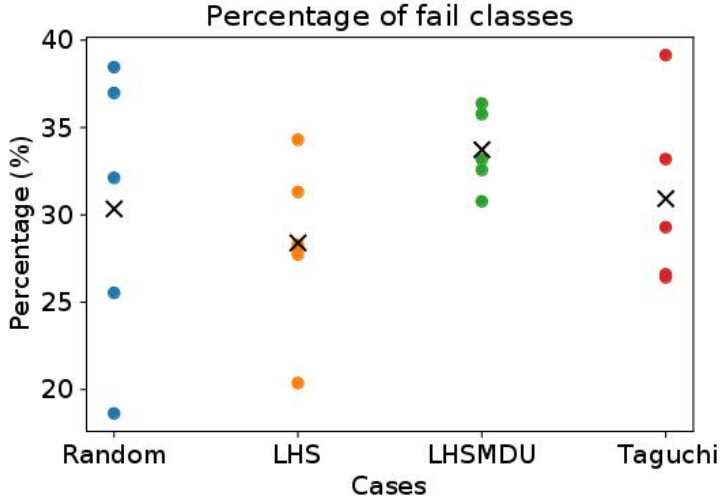
Taguchi method, OA of 128 samples, with strength 2, and factor level 2 for all variables

Initialization	**Tested parameters**	**Fails (%)**	**Changing (%)**	**Mute (%)**	**Pass (%)**
Random	23226.6	22.07	6.55	4.19	67.18
LHS	22183	20.24	6.55	3.42	69.8
LHSMDU	22578.8	21.19	6.77	3.32	68.72
Taguchi	**24440**	**22.85**	6.99	2.31	67.86

Stability with a smaller population

LHSMDU most stable, but not as good results as some runs of Random and Taguchi





Stability with a bigger population

Taguchi initialization is more stable with a larger population

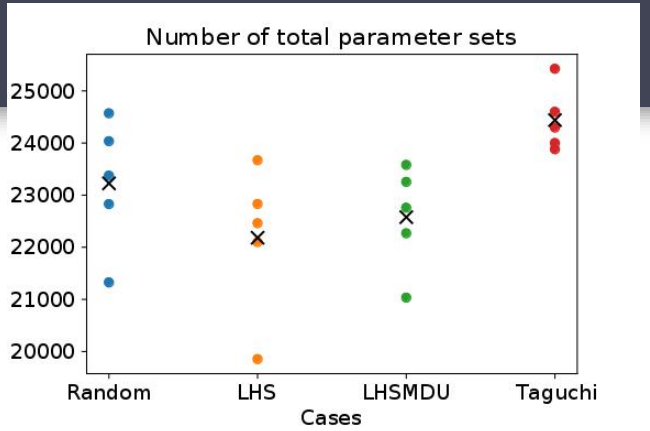
LHSMDU – a bit worse performance with larger population

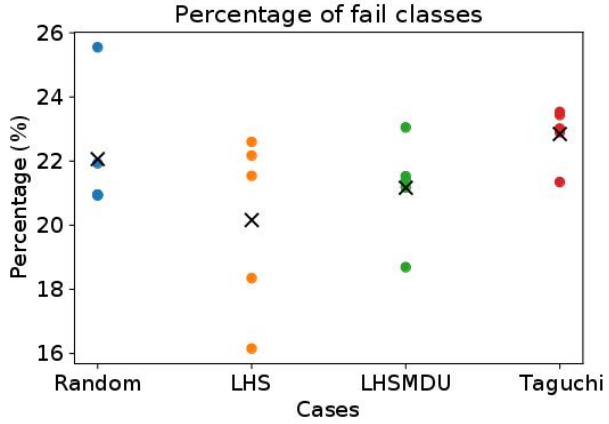
- Divides the intervals into the number of samples (population size)
- A lot of values get mapped to the same value

Random - the stability improved with a larger population

- We do not allow duplicate parameter sets

LHS - problem of the dimensionality





Conclusion

Memetic algorithm improved the results compared to random search

- 0.9% *fail* classes found by random search, 31.03% found with MA

On average, initialization techniques do not have a significant influence on the performance (considering the percentage of found *fail* classes)

The difference between the worst and best case for smaller population is 5.12%, while with the larger population it is 2.61%

Stability can be considered if we do not have time to run the algorithm multiple times

- LHSMDU and Taguchi are then preferred

Without great overhead, it is better to use sampling that promotes a well-distributed samples for the initial population, even more if a small population is used

Future work

This work is limited to one target, therefore the plan is to test the algorithm on different targets (e.g. running some encryption algorithm)

Distinguish between exploitable faulty outputs

Use orthogonal arrays with more factor levels, as it could improve the Taguchi method even more

Investigate which parameters would benefit more from a good distribution in the initial set

Thank you!

Questions?