Autotuning under Tight Budget Constraints: A Design of Experiments Approach

Pedro Bruel, Steven Quinito Masnada, Brice Videau, Arnaud Legrand, Jean-Marc Vincent, Alfredo Goldman

April 28, 2019

Outline

Autotuning: Optimizing Program Configurations

Architectures for High Performance Computing

How to write efficient code for each of these?

Autotuning

The process of automatically finding a configuration of a program that optimizes an objective

Configurations

- Program Configuration
  - Algorithm, block size, . . .
- Source code transformation
  - Loop unrolling, tiling, rotation . . .
- Compiler configuration
  - −O2, vectorization, . . .
- . . .

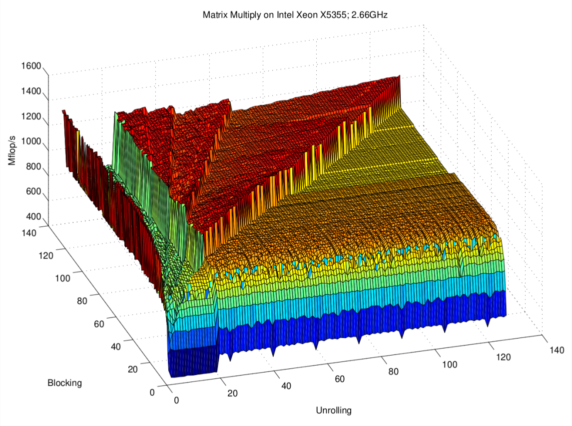
Objectives

- Execution time
- Memory & power consumption
- . . .

Search Spaces

Represent the effect of all possible configurations on the objectives

Can be difficult to explore, with multiple local optima and undefined regions



Unrolling, blocking and Mflops/s for matrix multiplication

Seymour K, You H, Dongarra J. A comparison of search heuristics for empirical code optimization. InCLUSTER 2008 Oct 1 (pp. 421-429)

Issue 1: Exponential Growth

Simple factors can generate large spaces:

- 30 boolean factors $\rightarrow 2^{30}$ combinations

Issue 2: Geometry

- Discrete or continous factors
- "Smoothness"
- Interactions between factors

Issue 3: Measurement Time

Time to compile:

- Benchmark GPU applications: 1~10s
- Benchmark FPGA applications: 1~10min
- Industrial FPGA applications: 1~10h

Autotuning: Multiple Approaches

Popular Approaches

- Exhaustive
- Meta-Heuristics
- Machine Learning

System	Domain	Approach
ATLAS	Dense Linear Algebra	Exhaustive
INSIEME	Compiler	Genetic Algorithm
Active Harmony	Runtime	Nelder-Mead
ParamILS	Domain-Agnostic	Stochastic Local Search
OPAL	Domain-Agnostic	Direct Search
OpenTuner	Domain-Agnostic	Ensemble
MILEPOST GCC	Compiler	Machine Learning
Apollo	GPU kernels	Decision Trees

Main Issues

- These approaches assume:
  - A large number of function evaluations
  - Seach space “smoothness”
  - Good solutions are reachable
- After optimizing:
  - Learn “nothing” about the search space
  - Can’t explain why optimizations work

Factors, Levels, Experiments & Designs

- Factors: program parameters
- Levels: possible factor values
- Experiment: setting each factor to a level
- Design: a selection of experiments to run

Analysis

Experiment results can be used to:

- Identify relevant parameters
- Fit a regression model

A small design for 7 2-level factors:

Run	A	B	C	D	E	F	G
1	1	-1	1	-1	-1	1	1
2	1	1	1	-1	1	-1	-1
3	-1	1	-1	-1	1	1	1
4	-1	1	1	1	-1	1	-1
5	1	-1	-1	1	1	1	-1
6	1	1	-1	1	-1	-1	1
7	-1	-1	1	1	1	-1	1
8	-1	-1	-1	-1	-1	-1	-1

Applying Design of Experiments to Autotuning

Our Approach

We are using:

- Efficient experimental designs to overcome issues related to exponential growth, geometry, and measurement time
- Analysis of variance to find relevant parameters
- User input to guide optimization

Design Requirements

- Support a large number of factors (Exponential Growth)
- Support numerical and categorical factors (Geometry)
- Minimize function evaluations (Measurement Time)

D-Optimal Designs

- Minimize variance of regression coefficient estimators
- Supports different factor types and numbers

Example

- Factors & Levels:

$$\mathbf{X} = (x_1 = (1, \ldots, 5), \\ x_2 = (\text{"}A\text{"}, \text{"}B\text{"}, \text{"}C\text{"}))$$

- Model: $\mathbf{Y} = \mathbf{X}\beta + \varepsilon$

Source code

```r
library(AlgDesign)

full_factorial <- gen.factorial(c(5, 3),
                     factors = c(2))

output <- optFederov(~ .,
                     full_factorial,
                     nTrials = 5)
```

Output

```
$D
[1] 0.5656854

$A
[1] 3.828125

$Ge
[1] 0.64

$Dea
[1] 0.57

$design
   1    5    6    10   13
X1 "-2" " 2" "-2" " 2" " 0"
X2 "1"  "1"  "2"  "2"  "3"

$rows
[1]  1  5  6 10 13
```

Linear Regression Model

The linear regression model:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k + \epsilon$$

We want to estimate $\beta_{0,\dots,k}$:

- Using $n > k$ observations $y_{1,\dots,n}$
- Distinct $x_{i1,\dots,ik},\ i = 1,\dots,n$

Experiments represented by:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_k x_{ik} + \epsilon_i$$

Ordinary Least Squares Estimator $\hat{\beta}$

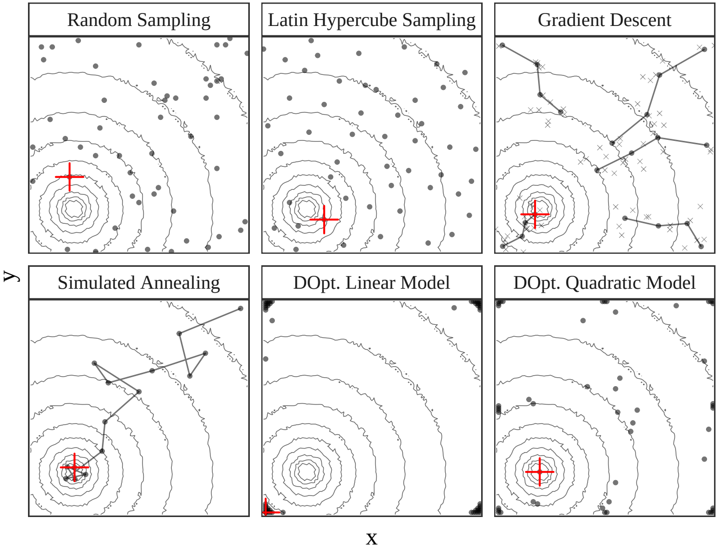
$$\hat{\beta} = (X^\intercal X)^{-1} X^\intercal Y$$

The variance of $\hat{\beta}$ is proportional to the covariance matrix $(X^\intercal X)^{-1}$

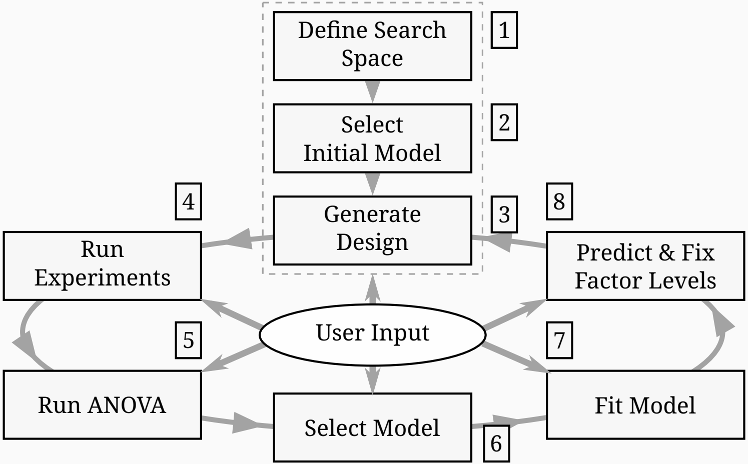
Design Criteria using $(X^\intercal X)^{-1}$

- D: determinant, minimizes generalized variance of $\hat{\beta}$
- A: trace, average variance of $\hat{\beta}$

Sampling Strategies



A Design of Experiments Approach to Autotuning



GPU Laplacian Kernel: A Motivating Example

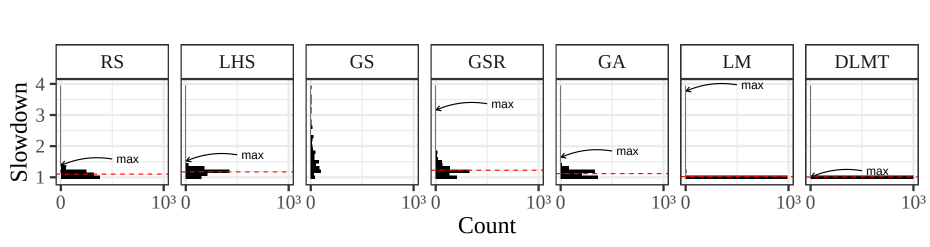
The Search Problem

- Relatively small valid search space
- Completely evaluated
- Known global optimum
- Known model approximation
- Budget of 125 points

Initial Model

$$\begin{aligned} cost = {} & y\_component\_number + 1/y\_component\_number + \\ & vector\_length + lws\_y + 1/lws\_y + \\ & load\_overlap + temporary\_size + \\ & elements\_number + 1/elements\_number + \\ & threads\_number + 1/threads\_number \end{aligned}$$

Results



GPU Laplacian Kernel: A Motivating Example

	Mean	Max
RS	120.00	125.00
LHS	98.92	125.00
GS	22.17	106.00
GSR	120.00	120.00
GA	120.00	120.00
LM	119.00	119.00
DLMT	54.84	56.00

Table 1: Points used by applications

Summary

Our approach:

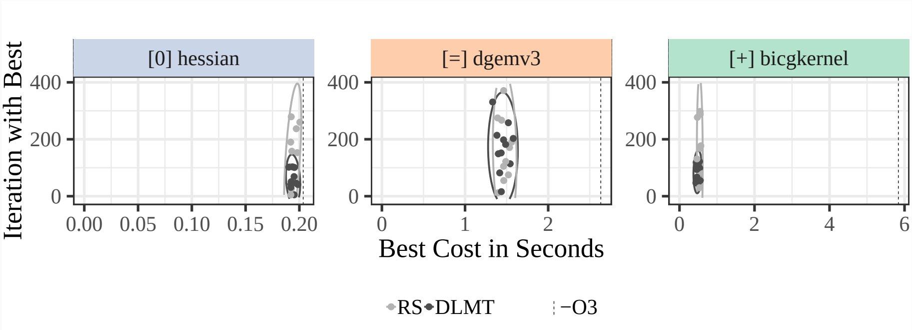
- Was always close to the optimum
- Used half of the budget

SPAPT: Search Problems in Automatic Performance Tuning

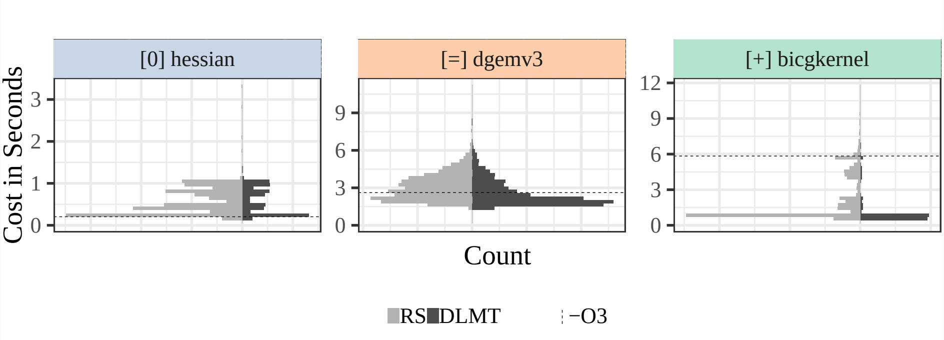
Kernel	Operation	Factors	Size
`atax`	Matrix transp. & vector mult.	18	2.6×10^{16}
`dgemv3`	Scalar, vector & matrix mult.	49	3.8×10^{36}
`gemver`	Vector mult. & matrix add.	24	2.6×10^{22}
`gesummv`	Scalar, vector, & matrix mult.	11	5.3×10^{9}
`hessian`	Hessian computation	9	3.7×10^{7}
`mm`	Matrix multiplication	13	1.2×10^{12}
`mvt`	Matrix vector product & transp.	12	1.1×10^{9}
`tensor`	Tensor matrix mult.	20	1.2×10^{19}
`trmm`	Triangular matrix operations	25	3.7×10^{23}
`bicg`	Subkernel of BiCGStab	13	3.2×10^{11}
`lu`	LU decomposition	14	9.6×10^{12}
`adi`	Matrix sub., mult., & div.	20	6.0×10^{15}
`jacobi`	1-D Jacobi computation	11	5.3×10^{9}
`seidel`	Matrix factorization	15	1.3×10^{14}
`stencil3d`	3-D stencil computation	29	9.7×10^{27}
`correlation`	Correlation computation	21	4.5×10^{17}

Balaprakash P, Wild SM, Norris B. SPAPT: Search problems in automatic performance tuning. Procedia Comp. Sci. 2012 Jan 1;9:1959-68.

SPAPT: Preliminary Results



SPAPT: Preliminary Results



Experimental Settings

- Using the same model for all applications
- Fixed number of iterations
- Automated approach

Summary

- Performance similar to random sampling
- Using less points

Summary

Our approach uses:

- Efficient experimental designs to overcome issues related to exponential growth, geometry, and measurement time
- Analysis of variance to find relevant parameters
- User input to guide optimization

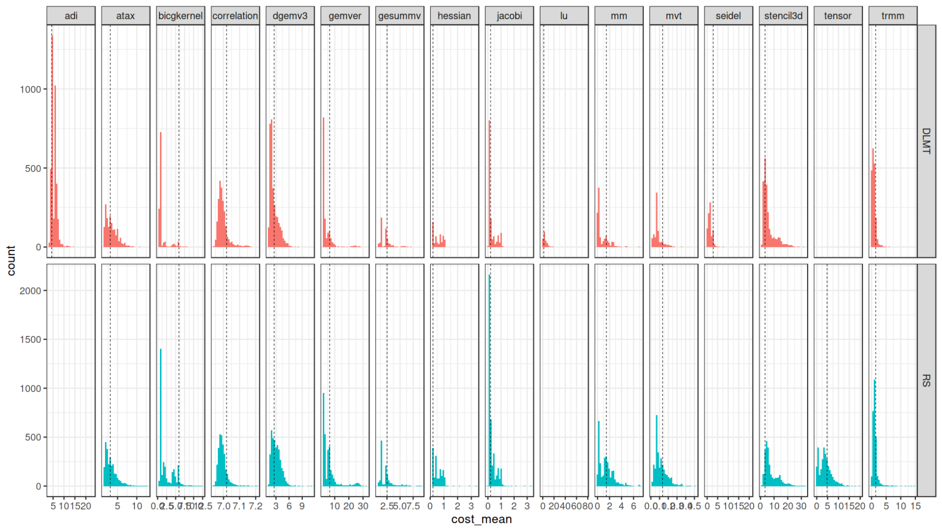
Perspectives

- Explore tailored models for each application
- Leverage user input and analysis
- Use our approach to autotune industrial-level FPGA applications

Autotuning under Tight Budget Constraints: A Design of Experiments Approach

Pedro Bruel, Steven Quinito Masnada, Brice Videau, Arnaud Legrand, Jean-Marc Vincent, Alfredo Goldman
April 28, 2019

SPAPT: Preliminary Results



Linear Regression Model

A simple regression model:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k + \epsilon$$

We want to estimate $\beta_{0,\ldots,k}$:

- Using $n > k$ observations $y_{1,\ldots,n}$
- Distinct $x_{i1,\ldots,ik}$, $i = 1, \ldots, n$

We will use n experiments such as:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_k x_{ik} + \epsilon_i$$

Least Squares Method

Writing in matrix form we get:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

The least squares method aims to minimize:

$$L = \sum_{i=1}^{n} \epsilon_i^2 = \boldsymbol{\epsilon}^\intercal \boldsymbol{\epsilon} = (\mathbf{Y} - X\boldsymbol{\beta})^\intercal (\mathbf{Y} - X\boldsymbol{\beta}) =$$

$$= Y^\intercal Y \; \boxed{-\beta^\intercal X^\intercal Y - Y^\intercal X\beta} + \beta^\intercal X^\intercal X\beta =$$

$$= Y^\intercal Y \; \boxed{-2\beta^\intercal X^\intercal Y} + \beta^\intercal X^\intercal X\beta$$

Minimizing Least Squares

The least squares method aims to minimize:

$$L = Y^\intercal Y - 2\beta^\intercal X^\intercal Y + \beta^\intercal X^\intercal X\beta$$

Derivative with respect to β, evaluated at $\hat{\beta}$:

$$\left.\frac{\partial L}{\partial \beta}\right|_{\hat{\beta}} = -2X^\intercal Y + 2X^\intercal X\hat{\beta} = 0$$

Where $\hat{\beta}$ is an estimator of β

Computing $\hat{\beta}$

The previous equation simplifies to:

$$\hat{\beta} = \left(X^\intercal X\right)^{-1} X^\intercal Y$$

The estimator $\hat{\beta}$ is proportional to $\left(X^\intercal X\right)^{-1}$

Dispersion or Covariance Matrix

- Information matrix: $X^\intercal X$
- Dispersion or Covariance matrix: $\left(X^\intercal X\right)^{-1}$

Design Efficiency: The Dispersion Matrix

Computing $(X^\intercal X)^{-1}$

A design $D_{n,2}$, with 2-level factors, will have a 3×3 dispersion matrix, if we assume linear relationships and no factor interactions:

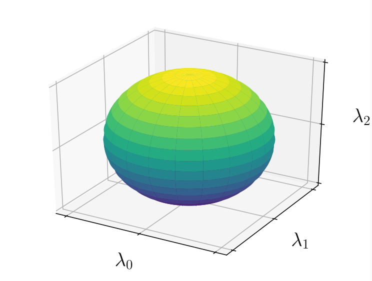
```
factorial <- gen.factorial(c(2, 2))
model <- model.matrix(~., factorial)
dispersion <- t(model) %*% model
eigen(dispersion)$values
```

```
            (Intercept) X1 X2
(Intercept)           4  0  0
X1                    0  4  0
X2                    0  0  4
```

```
[1] 4 4 4
```

Interpreting Eigenvalues of $(X^\intercal X)^{-1}$

The eigenvalues $\lambda_{0,1,2}$ of the dispersion matrix can represent its "size":



We can minimize the coefficient estimator $\hat{\beta}$ by minimizing the eigenvalues of $(X^\intercal X)^{-1}$

Design Efficiency: Metrics

Defining a Design

Consider a design $D_{n,k-1}$:

- $x_{1,\ldots,k-1}$ 2-level factors
- n experiments

Its $k \times k$ dispersion matrix $(X^\top X)^{-1}$:

- Constructed using the linear model:
  - $Y = \beta X + \epsilon$
- With eigenvalues $\lambda_{0,\ldots,m}$

We can define efficiency metrics for β based on the eigenvalues of the dispersion matrix

Some Efficiency Metrics based on $(X^\top X)^{-1}$

A-Efficiency

$$A_{eff} = \left(n \times \operatorname{tr}\left((X^\top X)^{-1}\right) / k\right)^{-1}, \; A_{eff} \in [0, 1]$$

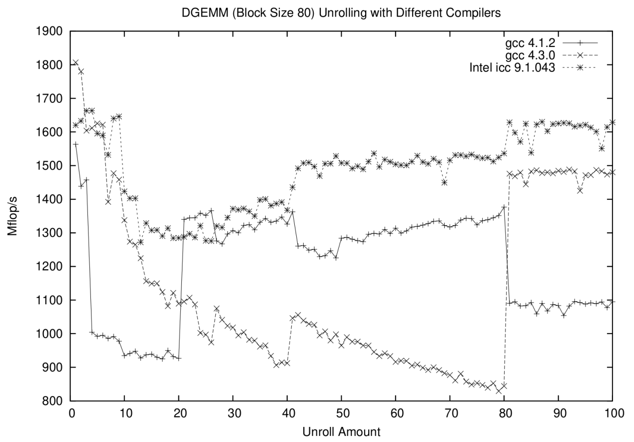
"Arithmetic mean" of eigenvalues of $(X^\top X)^{-1}$

D-Efficiency

$$D_{eff} = \left(n \times \left|(X^\top X)^{-1}\right|^{1/k}\right)^{-1}, \; D_{eff} \in [0, 1]$$

"Geometric mean" of eigenvalues of $(X^\top X)^{-1}$

Autotuning: Search Spaces



Compiler impact on performance

Seymour K, You H, Dongarra J. A comparison of search heuristics for empirical code optimization. InCLUSTER 2008 Oct 1 (pp. 421-429)

Applying Design of Experiments to Autotuning

Our Approach

Using efficient experimental design to overcome issues related to exponential growth, geometry, and measurement time

Design Requirements

- Support a large number of factors (Exponential Growth)
- Support numerical and categorical factors (Geometry)
- Minimize function evaluations (Measurement Time)

Main Design Candidates

Screening designs:

- Estimate main effects
- Aim to minimize runs
- Assume interactions are negligible

Mixed-Level designs:

- Factors have different numbers of levels
- Many optimality criteria

A Plackett-Burman screening design for 7 2-level factors:

Run	A	B	C	D	E	F	G
1	1	-1	1	-1	-1	1	1
2	1	1	1	-1	1	-1	-1
3	-1	1	-1	-1	1	1	1
4	-1	1	1	1	-1	1	-1
5	1	-1	-1	1	1	1	-1
6	1	1	-1	1	-1	-1	1
7	-1	-1	1	1	1	-1	1
8	-1	-1	-1	-1	-1	-1	-1

Screening Designs

Plackett-Burman designs for 2-level factors:

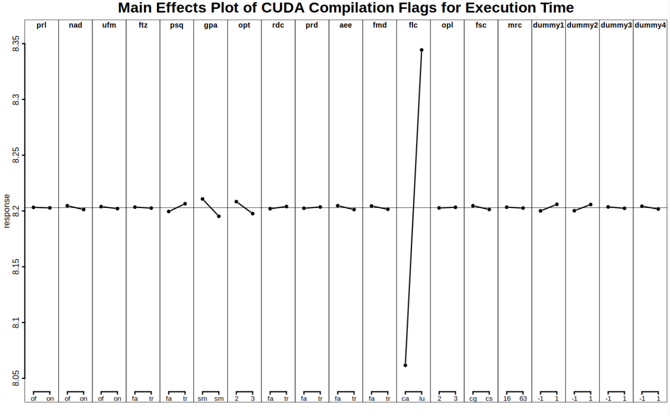
- Orthogonal arrays of strength 2
- Estimate the main effects of n factors with $n+1$ runs

Construction:

- For $n+1$ multiple of 4
- Identical to a fractional design if $n+1$ is a power of two

CUDA Compiler Flags

- Rodinia benchmark
- 15 factors, few with multiple levels
- 10^6 combinations
- 1~10s to measure
- Screening experiment:
  - 15 "2-level" factors
  - 4 "dummy" factors



A multi-level design for 1 2-level factor and 3 3-level factors:

Run	A	B	C	D
1	1	1	1	3
2	1	1	2	1
3	1	1	3	2
4	1	2	1	2
5	1	2	2	3
6	1	2	3	1
7	1	3	1	1
8	1	3	2	2
9	1	3	3	3
10	2	1	1	1
11	2	1	2	2
12	2	1	3	3
13	2	2	1	3
14	2	2	2	1
15	2	2	3	2
16	2	3	1	2
17	2	3	2	3
18	2	3	3	1

Mixed-Level Designs

Strategy 1: Contractive Replacement

- Find specific sets of k-level columns of a design, contract the set into a new factor of with more levels
- Maintain orthogonality of the design

Strategy 2: Direct Construction

Directly generate small mixed-level designs by solving Mixed Integer Programming problems

Strategy 3: D-Optimal Designs

Looking at Data: FPGA Compiler Parameters

FPGA Compiler Parameters

- CHStone benchmark
- 141 factors, most with multiple levels
- 10^{128} combinations
- 1~10min to measure
- Multiple objectives
- Search with meta-heuristics:
  - Unstructured data difficults analysis
  - We are working on obtaining more data