Open Access | https://doi.org/10.25080/uaha7744

SciPy 2025

July 7 - July 13, 2025

Proceedings of the 24th Python in Science Conference
ISSN: 2575-9752

OptiMask: Efficiently Finding the Largest NaN-Free Submatrix

Cyril Joly[1]

[1]Airparif

Abstract

When working with tabular data, many processes cannot handle missing values. While imputation strategies are commonly used, another approach is to extract a NaN-free submatrix. Simply dropping every row or column containing NaN values can drastically reduce the dataset, potentially leaving no usable data. OptiMask is a heuristic designed to solve the following optimization problem: identifying the submatrix of maximum size (in terms of the number of elements) without missing values. This submatrix is composed of rows and columns that are not necessarily contiguous in the original data. OptiMask determines the sets of rows and columns to remove from the input matrix, providing either an exact or near-optimal submatrix.

Keywords Optimization, Missing data, Machine Learning

1. Introduction

Missing data is a common challenge in data analysis, often represented as NaN (Not a Number) values in matrices or DataFrames. Many algorithms and statistical methods require complete datasets, necessitating effective handling of missing values [1], [2]. Traditional approaches include imputation (replacing missing values with estimates) and complete-case analysis (discarding rows/columns with any NaN) [3], [4]. However, imputation can introduce bias [5], while complete-case analysis may discard excessive data, especially when missing values are evenly distributed [6].

An alternative strategy is to identify the largest possible NaN-free submatrix, while preserving as much of the original, unaltered data as possible. This reduces to an optimization task: remove the minimal set of rows and columns to yield a NaN-free submatrix of maximum size.

This problem is computationally challenging, as the search space grows exponentially with the number of rows and columns containing NaN. Exact solutions (e.g., linear programming) guarantee optimality but are prohibitively expensive for large matrices.

Heuristic methods like OptiMask provide near-optimal solutions efficiently. OptiMask iteratively permutes rows and columns to isolate NaN values along a frontier, simplifying the search for the largest contiguous NaN-free submatrix. By combining randomization with multiple restarts, it reliably finds high-quality solutions. This paper explores the OptiMask algorithm, theoretical foundations, and practical performance across diverse datasets, including large and structured matrices. It also discusses the `optimask` Python package (https://pypi.org/project/optimask/), which enables applying the algorithm to matrix-like data structures popular with Python programmers.

Published Jul 10, 2025

Correspondence to
Cyril Joly
cyril.joly@airparif.fr

2. Problem Formalization and Challenges

2.1. Mathematical Definition

Given an $m \times n$ matrix A with missing values (NaN), the goal is to find:

- A subset of rows $\mathcal{R} \subseteq \{1, ..., m\}$ and columns $\mathcal{C} \subseteq \{1, ..., n\}$ such that the submatrix $A[\mathcal{R}, \mathcal{C}]$ contains no NaN values.
- The solution maximizing the size $| \mathcal{R} | \times | \mathcal{C} |$ of the submatrix.

2.2. The Fundamental Trade-off

When handling missing values in a matrix, we must decide whether to remove affected rows, columns, or a combination of both. The optimal choice depends on the matrix's dimensions and the distribution of NaN values:

1. **Toy Example**:
    - In **tall matrices** (rows > columns), removing a problematic **row** typically preserves more data.
    - In **wide matrices** (columns > rows), removing an affected **column** is usually preferable.

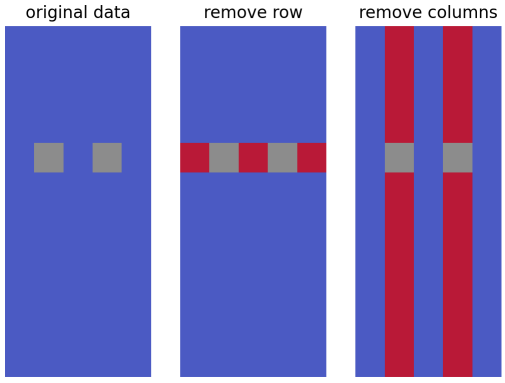

Figure 1. *In this toy example, removing the row with the two NaNs yields a larger submatrix (11x5) than removing the two columns, which results in a smaller submatrix (12x3) from the original 12x5 matrix.*

1. **Generalizing the Intuition**: The toy example suggests a greedy heuristic:
    - If a row contains many NaNs, removing that single row may be better than removing all corresponding columns (which could discard more data).
    - Conversely, if a column contains many NaNs, removing it might be better than eliminating all affected rows.
2. **Complex Cases Require a Systematic Approach**:
    - When NaNs are spread across both rows and columns, the optimal solution isn't obvious. Sometimes, the best solution involves a mix of row and column removals.
    - This necessitates an algorithmic strategy to maximize the preserved submatrix.

2.3. Linear Programming Formulation

The problem can be formulated using integer linear programming [7], defining decision variables for removing rows, columns, and individual cells, subject to constraints ensuring

all NaN values are handled. The objective is to minimize the total number of effectively removed cells, equivalent to maximizing the area of the remaining NaN-free submatrix.

Given:

- A matrix A of shape $m \times n$ with elements $a_{i,j}$.
- Decision variables for $i \in \{1, ..., m\}$ and $j \in \{1, ..., n\}$:
  - $r_i \in \{0, 1\}$: 1 if row i is removed, 0 otherwise.
  - $c_j \in \{0, 1\}$: 1 if column j is removed, 0 otherwise.
  - $e_{i,j} \in \{0, 1\}$: 1 if cell (i, j) is effectively removed (i.e., part of a removed row or column), 0 otherwise.

Constraints:

For all $i \in \{1, ..., m\}$ and $j \in \{1, ..., n\}$:

- If $a_{i,j}$ is NaN, then $e_{i,j} = 1$.
- $r_i + c_j \geq e_{i,j}$.
- $e_{i,j} \geq r_i$.
- $e_{i,j} \geq c_j$.

Objective Function:

Minimize the total number of effectively removed cells:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} e_{i,j} \tag{1}$$

This formulation can be solved using integer linear programming solvers (e.g., GLPK [8], Gurobi [9], CPLEX [10]), often interfaced via modeling languages like Pyomo [11] or PuLP [12] in Python for data science practitioners. However, its primary disadvantage is computational cost: for an $m \times n$ matrix, the formulation uses $m \times n + m + n$ binary variables, which becomes prohibitive for large matrices.

3. Algorithm

OptiMask is a heuristic designed to provide near-approximations of the optimal solutions to the problem. The core idea is to compute row and column permutations such that the search for the largest non-contiguous NaN-free submatrix reduces to finding a contiguous one.

3.1. Core Approach

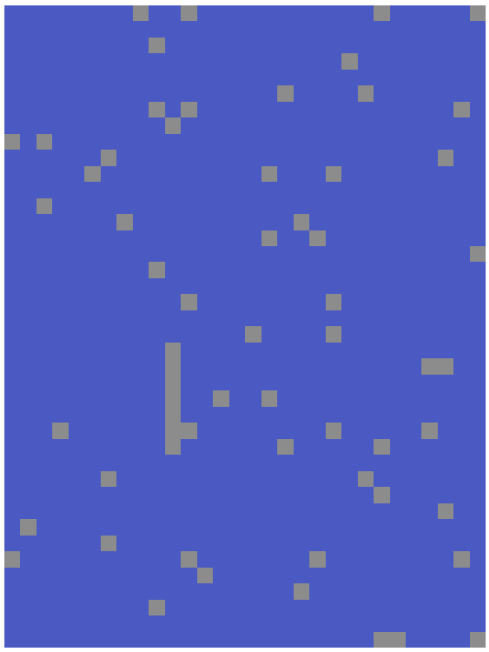

Figure 2. *An example 40x30 matrix, containing 63 missing values, illustrating the algorithm's steps. Grey cells represent missing values.*

OptiMask employs an iterative permutation-based algorithm to identify the largest NaN-free submatrix through these key steps:

1. **Problem Reduction**:
    - Isolate rows and columns containing at least one NaN value (rows or columns without NaNs are preserved, as there is no reason to remove them).
    - Create a boolean mask matrix where True represents NaN positions.
2. **Frontier Detection**:
    - Compute `hx`: column-wise, NaN cell with highest row index (i.e., furthest bottom).
    - Compute `hy`: row-wise rightmost NaN index (from the left).
    - These define the current "NaN frontier" of the matrix.

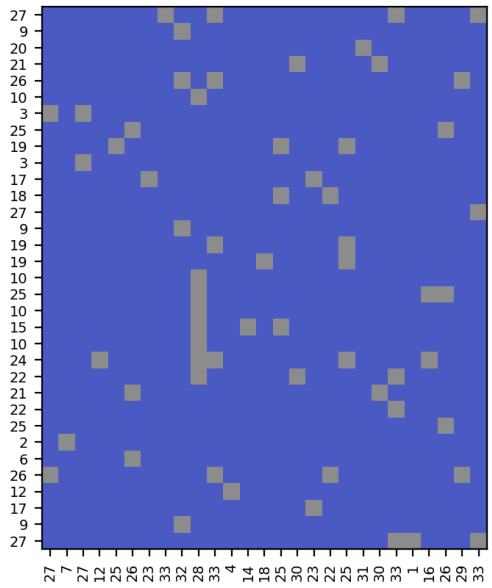

Figure 3. *Steps #1 and #2: isolating the 33 rows and 27 columns with NaNs and computing* `hx` *and* `hy`*.*

1. **Permutation Phase**:
    - Alternately sort rows and columns to push NaN values toward a Pareto frontier.
    - Even iterations: Sort columns by descending `hx`.
    - Odd iterations: Sort rows by descending `hy`.
    - Track all permutations applied during this process.
    - Repeat until both `hx` and `hy` form non-increasing sequences, indicating an optimal NaN frontier.

Figure 4. *Three alternate permutations lead to a Pareto frontier of NaNs.*

1. **Submatrix Extraction**:
    - Identify the largest contiguous NaN-free rectangle in the permuted space.

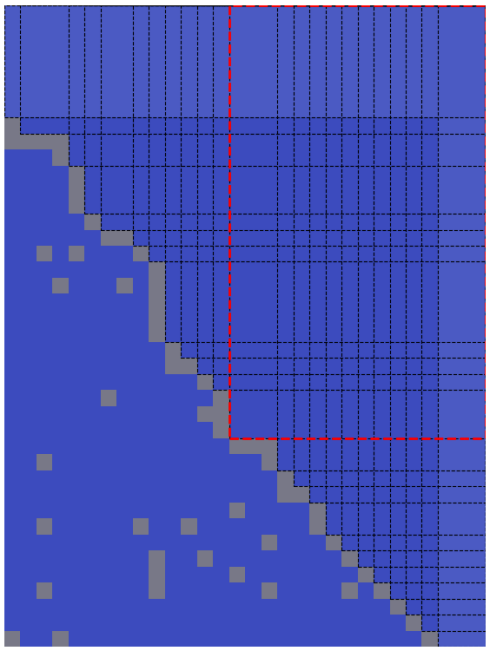

Figure 5. *OptiMask result in permuted space: the black-dotted rectangles are candidates for the largest contiguous NaN-free submatrix, with the red-dotted one selected for its maximal area.*

- Apply inverse permutations to map back to original row/column indices.

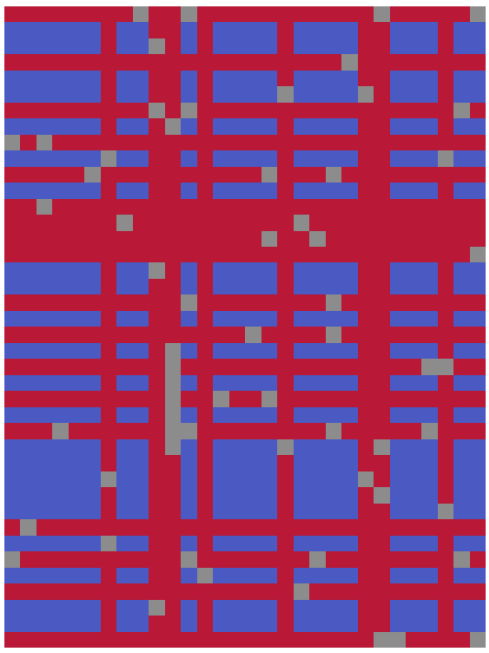

Figure 6. *OptiMask result: grey indicates missing values, red indicates removed rows and columns, blue marks the computed NaN-free submatrix. The algorithm computes a 27x17 submatrix, whereas discarding every row with NaN would result in a 7x30 submatrix, and discarding every column with NaN would yield a 40x3 submatrix.*

4. Python Package

A Python implementation of the algorithm is available on PyPI (https://pypi.org/project/optimask/) and conda-forge (https://anaconda.org/conda-forge/optimask). The library uses Numba [13] for speed and supports common input formats, including NumPy arrays [14], pandas DataFrames [15], and Polars DataFrames [16].

4.1. Basic Usage

```
import numpy as np
from optimask import OptiMask
from optimask.utils import generate_mar

# Generate a Missing At Random matrix with 2% NaN values
x = generate_mar(m=100_000, n=1_000, ratio=0.02, rng=0)
rows, cols = OptiMask(random_state=0).solve(x)
np.isnan(x[np.ix_(rows, cols)]).any()  # False
len(rows), len(cols)  # (37386, 49)
```

This computation takes approximately 200ms on an average personal computer. The implementation provides the sorted indices of the rows and columns to retain, ensuring that the relative order of the elements is preserved.

4.2. Handling Missing Data for Machine Learning

OptiMask removes missing values (NaN) from datasets while maximizing usable data. It does this by finding an optimal subset of samples (rows) and features (columns) to discard, ensuring the remaining data contains no missing values. This makes the dataset directly usable for machine learning models that require complete data, such as scikit-learn's linear models:

```
import numpy as np
from optimask import OptiMask
from sklearn.datasets import make_spd_matrix
from sklearn.linear_model import LinearRegression

def load_data_with_nan(m, n, nan_ratio):
    mean = np.random.randn(n + 1)
    cov = make_spd_matrix(n + 1)
    data = np.random.multivariate_normal(mean=mean, cov=cov, size=m)
    mask = np.random.rand(*data.shape) < nan_ratio
    data[mask] = np.nan
    return data[:, :-1], data[:, -1]

# Simulate a dataset with missing values
X, y = load_data_with_nan(m=10_000, n=100, nan_ratio=0.02)

# Drop samples where the target (y) is missing
valid_samples = np.isfinite(y)
X, y = X[valid_samples], y[valid_samples]

# Apply OptiMask to remove remaining NaNs
rows, cols = OptiMask().solve(X)
X_clean, y_clean = X[np.ix_(rows, cols)], y[rows]
print(X_clean.shape, y_clean.shape) # (3581, 50) (3581,)

# Train a model on the NaN-free data
model = LinearRegression().fit(X=X_clean, y=y_clean)
```

By strategically selecting which rows and columns to keep, OptiMask ensures the dataset remains meaningful while becoming fully trainable.

5. Conclusion

OptiMask provides a heuristic for finding the largest NaN-free submatrix in large datasets where exact methods like linear programming become computationally impractical. By strategically permuting rows and columns to isolate missing values, it offers a practical solution that preserves maximal data without imputation. The Python implementation supports common data structures (numpy, pandas, polars) and delivers results efficiently even for large matrices.

Future work will explore theoretical guarantees on the approximation quality and extensions to weighted optimization problems. The implementation will continue to be optimized for speed in subsequent versions. Finally, an OptiMask-based algorithm for tabular imputation will be developed and benchmarked against MICE (Multiple Imputation by Chained Equations, [5]) to evaluate whether it can achieve better or faster results.

Acknowledgements

This work was funded by Airparif. I'd like to thank Paul Catala (Université de Lorraine) and Alexis Lebeau (RTE) for their assistance and review.

References

[1] R. J. Little and D. B. Rubin, *Statistical analysis with missing data*, vol. 793. John Wiley & Sons, 2019. doi: 10.1002/9781119482260.

[2] D. B. Rubin, *Multiple imputation for nonresponse in surveys*, vol. 81. John Wiley & Sons, 2004. doi: 10.1002/9780470316696.

[3] J. L. Schafer, *Analysis of incomplete multivariate data*. CRC press, 1997. doi: 10.1201/9781439821862.

[4] S. Van Buuren, *Flexible imputation of missing data*. CRC press, 2018. doi: 10.1201/9780429492259.

[5] I. R. White, P. Royston, and A. M. Wood, "Multiple imputation using chained equations: issues and guidance for practice," *Statistics in medicine*, vol. 30, no. 4, pp. 377–399, 2011, doi: 10.1002/sim.4067.

[6] C. K. Enders, *Applied missing data analysis*. Guilford press, 2010. doi: 10.1111/j.1467-842X.2012.00656.x.

[7] L. A. Wolsey, *Integer programming*. John Wiley & Sons, 2020. doi: 10.1002/9781119606475.

[8] A. Makhorin, "GLPK (GNU linear programming kit)," 2012. [Online]. Available: http://www.gnu.org/software/glpk/

[9] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual." [Online]. Available: https://www.gurobi.com/

[10] IBM ILOG CPLEX, "V12. 1: User's Manual for CPLEX," 2009.

[11] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: modeling and solving mathematical programs in Python," *Mathematical Programming Computation*, vol. 3, no. 3, pp. 219–260, 2011, doi: 10.1007/s12532-011-0026-8.

[12] S. Mitchell, M. OSullivan, and I. Dunning, "PuLP: a linear programming toolkit for python." [Online]. Available: https://pythonhosted.org/PuLP/

[13] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 2015, pp. 1–6. doi: 10.1145/2833157.2833162.

[14] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020, doi: 10.1038/s41586-020-2649-2.

[15] W. McKinney and others, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, 2010, pp. 51–56. doi: 10.25080/Majora-92bf1922-00a.

[16] R. Vink, "Polars: Lightning-fast DataFrame library for Rust and Python." [Online]. Available: https://pola.rs/