SoftwareX 36 (2026) 103194

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Trasgu: A wrapper for simplified vine copula fitting with Chimera across local and high-performance computing platforms☆

Valvanuz Fernández-Quiruelas[a,*], Patricia Mares-Nasarre[b], Cornelis Marcel Pieter 't Hart[b,c], Oswaldo Morales-Nápoles[b]

[a] Departamento de Matemática, Aplicada y Ciencias de la Computación, Universidad de Cantabria, Santander, Spain
[b] Department of Hydraulic Engineering, Faculty of Civil Engineering and Geosciences, Delft University of Technology, Stevinweg 1, 2628 CN Delft, the Netherlands
[c] Haskoning Nederland, PO Box 1132, Amersfoort, the Netherlands

ARTICLE INFO

Keywords:
Vine copulas
Regular vines
High-performance computing
Scientific workflows
Chimera

ABSTRACT

Vine copulas represent multivariate probability distributions through sequences of trees whose edges are modelled with bivariate copulas. Chimera is a catalogue developed at TU Delft that contains all possible regular-vine tree sequences with four to eight nodes, represented by 663,206,904 matrices. Exhaustively fitting these matrices provides a reference for evaluating structure-selection algorithms based on heuristics, but requires managing millions of independent fits. Trasgu is an open-source Python package developed to make such studies reproducible on personal computers and high-performance computing systems. We illustrate the use of Trasgu through a simulation study and a real-world application with field data on ship-induced waves.

Metadata

The ancillary data in the See Code metadata table describe the software release used in this article.

Code metadata.

Nr.	Code metadata description	Value
C1	Current code version	v1.0.0
C2	Permanent GitHub link to code/repository used for this code version	https://github.com/SantanderMetGroup/trasgu/tree/v1.0.0
C3	Legal Code License	MIT License.
C4	Code versioning system used	Git.
C5	Software code languages, tools, and services used	Python; uv; Zarr; pyvinecopulib; Snakemake; SLURM; GitHub Actions; PyPI.
C6	Compilation requirements, operating environments & dependencies	No compilation required; Linux, macOS, or Windows; Python 3.11–3.13; NumPy, PyYAML, Zarr, fsspec, pyvinecopulib, and Snakemake; optional SLURM executor plugin.
C7	If available Link to developer documentation/manual	https://santandermetgroup.github.io/trasgu/
C8	Support email for questions	valvanuz.fernandez@unican.es

☆ The trasgu is a mythological creature present in Galician, Asturian and Cantabrian traditional culture. When treated well, the trasgu does many small tasks quickly around the house during the night. Folklore says that when confronted with impossible tasks, like fitting 663,206,904 vine copulas to data, he will still try to solve them.

* Corresponding author.
Email address: valvanuz.fernandez@unican.es (V. Fernández-Quiruelas).

https://doi.org/10.1016/j.softx.2026.103194
Received 1 September 2026; Received in revised form 20 October 2026; Accepted 29 October 2026
Available online 4 November 2026
2352-7110/© 2026 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Motivation and significance

Simplified vine copulas are flexible models for representing multivariate dependence through bivariate building blocks [1–6]. In applications, however, the modeller has to choose a regular vine structure, the bivariate copula families and their parameters. In principle, one may fit all possible regular vines and select the best model according to AIC, another likelihood-based criterion, or formal hypothesis tests. In practice this is rarely done, because the number of regular vines grows extremely fast with the number of variables.

Most available software for vine copulas, including implementations in R and Python, relies on sequential or greedy algorithms to make the fitting problem computationally feasible [7,16,17]. This is often a necessary modelling choice, but it is also a *simplifying decision* in the sense discussed in [18]. The modeller not only adopts the usual simplifying assumption for conditional copulas, but also accepts a reduced search over the space of regular vines. The consequences of this decision can only be studied systematically when sufficiently large sets of regular vines are available and can be fitted to data.

Chimera was developed for this purpose. It contains all regular vine matrices on up to eight nodes: 24 on four nodes, 480 on five nodes, 23,040 on six nodes, 2,580,480 on seven nodes and 660,602,880 on eight nodes [9].

Recent work using Chimera has shown that complete brute-force fitting is possible with the aid of high-performance computing, and that such experiments can be used to test commonly used algorithms such as Dißmann's algorithm [12,13]. Morales-Nápoles [12] describes the use of Chimera and HPC for systematic experiments with AIC and Vuong's test [8], and discusses future research directions in risk and reliability analysis. The AIC simulation study shows the scale that can be reached: more than 1.4 billion vine copula models were fitted to synthetic and real-life data sets with three to eight variables.

Carrying out such studies requires repeated exhaustive evaluations across synthetic and empirical datasets. In the experiments reported in [9,12,13], high-dimensional fits required ad-hoc matrix partitioning and manual batch submission on a dedicated cluster. Consequently, this procedure was not readily reusable by researchers without specialist computing support, even on the same infrastructure. To overcome these limitations, Trasgu was developed as an open-source Python package that automates matrix partitioning, parallel execution, and result aggregation for researchers with suitable computing resources.

The aim of this article is to demonstrate how Trasgu orchestrates these exhaustive fitting studies in practice. The benchmarks and illustrative examples presented here focus on Trasgu's computational performance, workflow orchestration and practical use. For methodological details of the fitting studies and the domain-specific interpretation of their results, readers are referred to [9,12,13].

The necessity of Trasgu becomes apparent when evaluating a direct sequential loop that fits every matrix in the collection one by one. Throughout this article, the baseline configuration uses `pyvinecopulib` [16] with one-parameter copula families and AIC selection, and is included in all benchmarks and Dißmann comparisons. Under these settings, Fig. 1 compares the size of the Chimera collections with the estimated time of sequential fitting for 300 observations, measured on a single dedicated core of an Intel Xeon Gold 5218 CPU @ 2.30 GHz. The four- and five-variable collections were fitted in full. For six to eight variables, the mean fitting time of 1000 evenly spaced matrices per collection was multiplied by the collection size. The resulting times are less than one second for four variables, 32 s for five, 37 min for six, 4.1 days for seven and 3.8 years for eight. Consequently, a direct loop is practical up to five or six variables, whereas higher dimensions require a different approach.

Increasing the number of threads used by `pyvinecopulib` reduces the fitting time for an individual structure by distributing the computation across multiple CPU cores. However, the speedup quickly saturates.

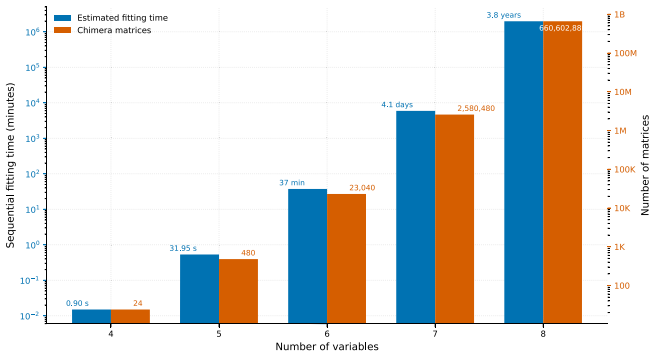


Fig. 1. Estimated sequential fitting time (blue; measured for four and five variables and extrapolated from 1,000 fits for six to eight variables) and number of Chimera matrices (Orange) for 300 observations, one CPU core, and the one-parameter families available in `pyvinecopulib`.

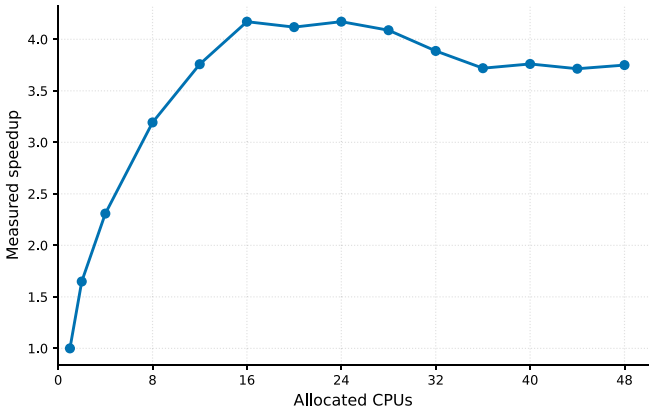


Fig. 2. Measured median speedup for one fixed eight-dimensional vine fit with 300 observations as the number of `pyvinecopulib` threads increases.

To measure this effect under controlled conditions, we selected one eight-dimensional regular vine structure and fitted it seven times at each CPU allocation (1 to 48 cores) in randomized order on the same Intel Xeon Gold 5218 system. Speedup was calculated from the median fitting times relative to one CPU. It reached approximately 4.2 with both 16 and 24 CPUs, showing little improvement over this range (Fig. 2). The benchmark scripts, synthetic inputs, raw measurements and plotting routines are provided in the repository's `benchmarks/` directory.

The scale of the eight-dimensional collection also creates a data-access challenge. Loading all 660,602,880 matrices as unsigned 8-bit integers requires 42.28 GB of memory for the array alone, exceeding the capacity of many personal computers and individual HPC job allocations. Practical use requires indexed, chunk-wise access so that only the matrix ranges needed for a given task are retrieved.

Trasgu divides the matrix collection into deterministic chunks, retrieves the corresponding matrix ranges, distributes independent fits among worker processes and combines their results while retaining Chimera identifiers. Trasgu supplies the configuration interface and workflow connecting these operations. Statistical fitting is performed by `pyvinecopulib`, while Snakemake [14] manages task dependencies, execution through local or HPC profiles, and resumption from valid completed outputs. This integration allows fitting campaigns to proceed without manually submitting successive batches of jobs.

2. Software description

Trasgu is a Python package for fitting the complete collection of regular-vine structures in Chimera corresponding to the dimensionality of a multivariate pseudo-observations dataset (input data). A run is defined by a `trasgu.yaml` configuration file, in which the user specifies the input data, fitting controls and computational resources. The resulting CSV file contains one row for each fitted Chimera regular-vine structure, reporting its matrix identifier, number of fitted parameters and AIC. These results allow the candidate structures to be compared and the structure with the lowest observed AIC to be identified.

2.1. Software architecture

Fig. 3 summarizes the architecture. The user provides a multivariate pseudo-observation dataset (input data) and the `trasgu.yaml` configuration file. The input data can be stored as whitespace-delimited text, CSV, TSV or a NumPy array; a subset of columns can be selected in the configuration. Trasgu validates the data, infers the number of variables from the input dataset (Trasgu supports between four and eight variables similar to Chimera) and selects the corresponding Chimera collection.

To support indexed access, we converted the original compressed Python pickle files [10] to chunked Zarr arrays [11]. Trasgu retrieves the matrix ranges assigned to each computational task from these arrays. Matrices are accessed on demand via HTTP from a server at the University of Cantabria, with no prior data download required. A local copy or an alternative remote URL can be specified through the `chimera_url` directive in `trasgu.yaml`.

Once Trasgu has determined the number of variables and has read the configured `chunk_size`, it partitions the corresponding Chimera collection into deterministic chunks, each processed as a separate computational task or job. For example, with seven variables and a chunk size of 20,000, the 2,580,480 structures are divided into 130 chunks: 129 contain 20,000 structures and the final chunk contains 480.

Each chunk runs as a separate task and fits its assigned structures. The user can specify the number of worker processes used to parallelize these fits through `max_workers` in `trasgu.yaml`. Trasgu uses Python's multiprocessing module to distribute the matrices within each chunk among the workers. Their results are collected and aggregated once all workers have completed their fits.

Each regular-vine structure is fitted independently using `pyvinecopulib`, which selects the pair-copula families and estimates their parameters for the specified structure. By default, Trasgu uses one-parameter families and AIC-based family selection. Non-default fitting controls can be provided as a serialized `FitControlsVinecop` object in `trasgu.yaml`.

This first level of parallelization may be sufficient for users who want to exploit the cores available on a personal computer or a cloud virtual machine, provided that individual chunks have manageable execution times. If a chunk takes several days to complete, an unexpected shutdown or restart would interrupt the task and require the entire chunk to be fitted again.

Large campaigns therefore require an orchestration layer that manages the complete set of chunks, submits new tasks as computational resources become available and keeps track of their progress. This is particularly important in HPC environments, where many chunks may be distributed across multiple compute nodes and remain queued until the requested resources are available. The orchestrator must identify completed and pending chunks, preserve valid results after an interruption and relaunch failed or incomplete tasks.

Rather than implementing a dedicated workflow manager, Trasgu delegates these responsibilities to Snakemake [14]. Snakemake orchestrates the chunk jobs, supports progress monitoring, permits dry runs and allows interrupted workflows to resume without repeating completed work. Failed jobs can also be relaunched automatically when retries are configured.

A further advantage of Snakemake is the separation of the scientific configuration from the computational infrastructure. Input data, fitting controls and chunk definitions remain unchanged, while infrastructure-specific settings—such as the execution backend, queue, concurrency limits, memory and runtime—are defined through Snakemake profiles. Users can therefore adapt the same workflow to a personal computer, a cloud virtual machine or an HPC system without changing the scientific specification of the analysis. This separation improves portability, facilitates reproducibility and allows different users to run the same analysis using the resources available at their institutions.

In this design, Trasgu integrates `pyvinecopulib` and Snakemake to organize exhaustive searches over Chimera collections. While `pyvinecopulib` fits individual vine copulas and Snakemake manages workflow execution, Trasgu partitions the matrix collection into deterministic chunks, retrieves the corresponding matrices from Zarr arrays, distributes fits among worker processes within each chunk, and consolidates the results while preserving their Chimera indices.

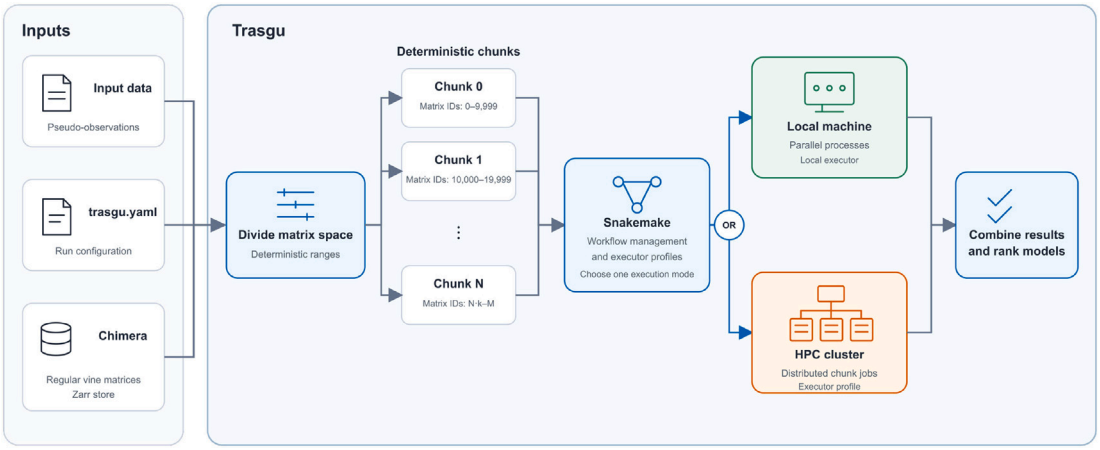


Fig. 3. Trasgu architecture. Input pseudo-observations and YAML configuration define a fitting campaign. Trasgu partitions the Chimera matrix collection into independent chunks. Snakemake tracks their state and executes missing targets either locally or on an HPC cluster using the selected profile, before the chunk-level results are combined.

2.2. Software functionalities

Each fitting campaign is organized in a self-contained directory and configured through `trasgu.yaml`. The user specifies the input data and may adjust the chunk size, number of worker processes and fitting controls. A minimal configuration is:

```
data_file: input6_300_clayton_high.txt
chunk_size: 1000
max_workers: 4
```

Before execution, `trasgu_count_chunks` reports the number of tasks. Users can then measure the execution time of one chunk with `trasgu_time_fit` and preview the planned jobs with `trasgu_run --dry-run`. This helps them select a suitable chunk size and worker count before committing computational resources.

Campaigns are started with `trasgu_run`, either locally or through the selected Snakemake profile for distributed execution. `trasgu_monitor` reports completed and pending chunks and their completion percentages. Relaunching an interrupted campaign schedules only the missing or outdated results, preserving valid chunk outputs.

Once the campaign is complete, `trasgu_combine` aggregates the chunk outputs into a CSV table containing the Chimera matrix identifier, number of fitted parameters and AIC. `trasgu_best_fits` identifies and exports the models with the lowest AIC. Additional commands support fitting an individual chunk or matrix, locating Chimera matrices, downloading the Zarr collections for offline use and copying packaged examples. A complete summary of the available command-line tools is provided in Appendix A; detailed options and usage examples are available in the online documentation.

2.3. Software verification and correctness

To verify the integrity of the Zarr storage, deterministic chunking and parallel execution, Trasgu was verified against sequential Python fits using the original TU Delft Chimera files. Two five-dimensional models shared Chimera structure 61: one used ten Clayton pair-copulas ($\theta = 3.1819$), and the other six Clayton ($\theta = 3.1819$) and four Gaussian ($\rho = 0.5$) pair-copulas. The AIC values produced by Trasgu matched the manual benchmark within numerical precision (maximum absolute difference below 5×10^{-7}). Complete configurations and reproducibility records are available in the repository's validation documentation.

3. Illustrative examples

All experiments reported in this section were executed on Altamira, the high-performance computing infrastructure of the University of Cantabria, operated at IFCA-CSIC as part of the Spanish Supercomputing Network. Trasgu submitted and monitored the computational jobs through Altamira's SLURM queueing system [15].

Two experiments illustrate the complete workflow, from input preparation and configuration to distributed fitting and result aggregation. The first uses synthetic data generated from a known seven-dimensional vine, allowing the exhaustive search to be evaluated under controlled conditions. The second applies the same workflow to an eight-dimensional ship-wave data set and demonstrates the scale that can be addressed on an HPC system. The configurations, workflow snapshots and compact results for both experiments are distributed with the software; the source ship-wake CSV is not redistributed, while the accompanying data record contains the larger execution artifacts (https://zenodo.org/records/23098246).

3.1. Seven-dimensional synthetic experiment

This experiment demonstrates the use of Trasgu to carry out repeated exhaustive searches of the seven-dimensional Chimera collection under controlled conditions. Using data generated from a known vine model, we assess how often the generating matrix is selected and compare the AIC of the exhaustive search with that of Dißmann's heuristic, while documenting the computational resources and execution time required.

Synthetic data were generated using the regular-vine structure (see Fig. 4) corresponding to the Chimera matrix 252,000, M_{gen}, which is one of the 2,580,480 seven-dimensional Chimera matrices:

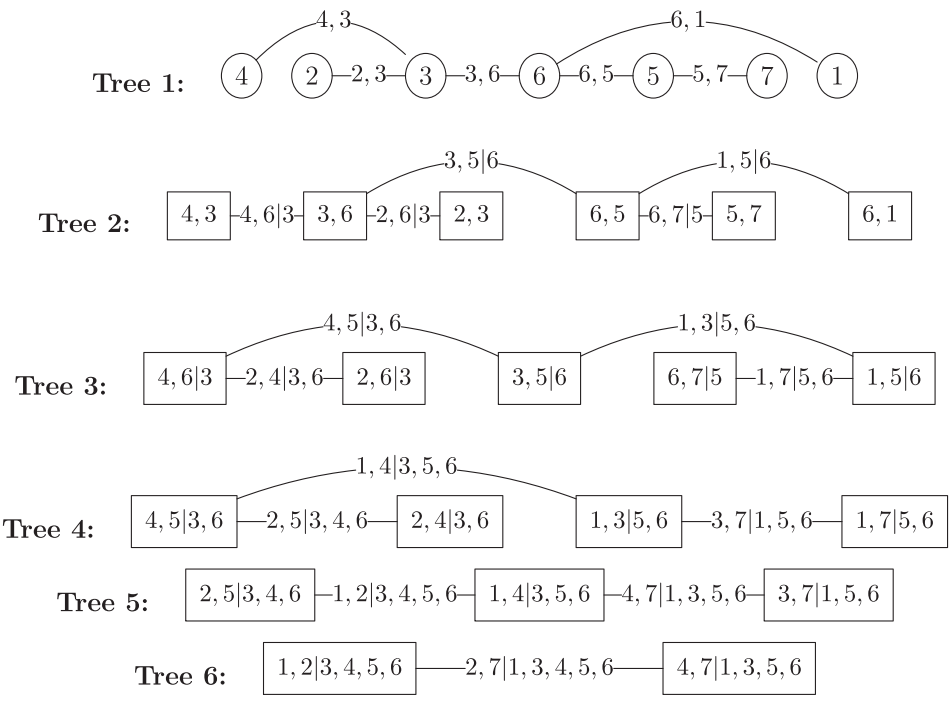


Fig. 4. Tree decomposition of the regular-vine structure M_{gen}.

$$M_{\text{gen}} = \begin{pmatrix} 5 & 6 & 3 & 3 & 6 & 5 & 5 \\ 6 & 5 & 6 & 6 & 5 & 6 & 0 \\ 1 & 3 & 4 & 5 & 3 & 0 & 0 \\ 3 & 4 & 5 & 4 & 0 & 0 & 0 \\ 4 & 2 & 2 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Every edge was assigned a Clayton copula with parameter 3.1819, corresponding to Kendall's $\tau \simeq 0.61$, to obtain a strongly dependent test case. 100 independent data sets of 300 observations were generated. For each data set, Trasgu fitted all 2,580,480 seven-dimensional Chimera matrices using the one-parameter copula families provided by `pyvinecopulib` and selected the model with the lowest AIC. For comparison, Dißmann's heuristic algorithm [7] was applied to the same synthetic observations. A constrained reference model was also fitted using the generating matrix and Clayton copulas, so that only the copula parameters were estimated.

The matrix space was divided into ten chunks of at most 260,000 matrices. Each chunk used up to 16 worker processes, while Snakemake submitted and monitored the chunk jobs through the SLURM executor profile. Once all chunks of a simulation were available, the workflow combined their CSV files and refitted the lowest-AIC candidate.

The generating matrix was selected in 60 of the 100 simulations; 17 different matrices were selected at least once. Both the exhaustive search and Dißmann's algorithm were executed using identical settings in `pyvinecopulib` (one-parameter candidate families, AIC selection criterion and maximum likelihood estimation). The exhaustive result had a lower AIC than the Dißmann's result in every simulation. As shown in Fig. 5, the median absolute AIC difference from the constrained Clayton reference was 52.2 for the exhaustive search and 1573.9 for Dißmann. As discussed in [12,13], such large discrepancies arise from the greedy nature of heuristic tree selection under asymmetric tail dependence, which locks into sub-optimal initial trees. Trasgu was developed specifically to make this type of large-scale comparative investigation computationally tractable and reproducible.

The 100 simulations generated 1000 fitting jobs on Altamira `compute` nodes equipped with Intel Xeon Platinum 8160 processors. Their mean effective duration was 17 min 50 s, and at most 60 jobs ran simultaneously, corresponding to 960 concurrent cores. The interval from the start of the first chunk to completion of the last was 5 h 9 min 49 s; these values exclude scheduler queueing time. Based on the mean job duration, running the same 100 jobs sequentially using a single core would have required approximately 198 days.

3.2. Application to ship-induced waves

To demonstrate Trasgu's capability to orchestrate an exhaustive search across a massive combinatorial space (660,602,880 structures) and recover from runtime interruptions on HPC clusters, we evaluated an observational dataset of ship-induced waves from the Savannah River [21]. Eight variables describing ship geometry and motion, hydrodynamic conditions and measured wake response were selected and transformed into pseudo-observations. From the field campaign records, 179 complete observations were retained by selecting vessel passages with concurrent measurements across all eight channels. Continuous marginals were transformed into uniform pseudo-observations on $(0, 1)$ via empirical rank normalization ($u_{ik} = \text{rank}(x_{ik})/(n+1)$) using `pyvinecopulib`'s default `to_pseudo_obs`, resolving any ties with average ranks.

The 660,602,880 eight-dimensional Chimera structures were partitioned into chunks of 4,000,000 matrices, producing 166 independent fitting jobs. The workflow was initially submitted to Altamira's `compute` partition, whose nodes use Intel Xeon Platinum 8160 processors, with eight cores per job. After 129 chunks had completed, execution was interrupted because the disk quota was exhausted. Once the storage problem had been resolved, the remaining 37 chunks were submitted to the `wncompute_ifca` partition with 32 cores per job to shorten their execution. The execution profile and worker allocation were changed, while the input data, matrix ranges, fitting controls and completed chunk outputs remained unchanged. On relaunch, the workflow detected the existing chunk outputs and did not recompute them.

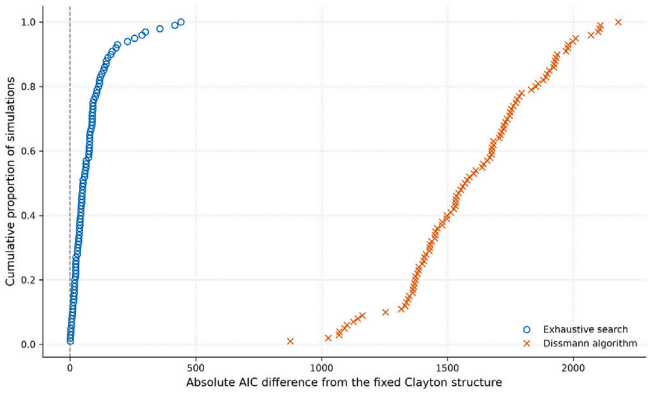


Fig. 5. Empirical cumulative distributions of the absolute AIC difference from a model fitted with the generating matrix and the Clayton family, across 100 independently simulated data sets with 300 observations.

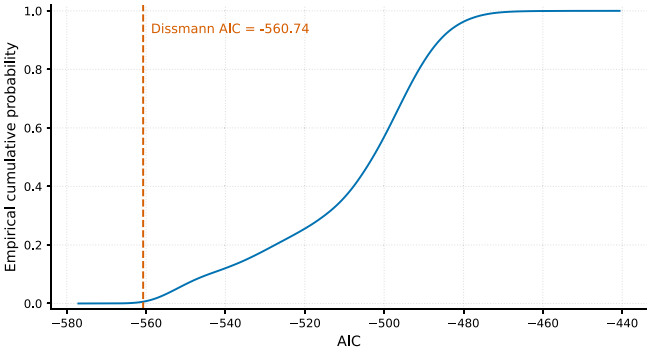


Fig. 6. Empirical cumulative distribution of AIC across all 660,602,880 eight-dimensional Chimera matrices fitted to the ship-wake data. The dashed line marks the AIC obtained with Dißmann's sequential method.

Excluding the approximately 47-hour interruption, the complete experiment required about 25 hours of active wall-clock time: 16 h 34 min for the initial phase and 8 h 24 min for the resumed phase and final aggregation.

The empirical AIC distribution is shown in Fig. 6. At the Dißmann result, the empirical cumulative probability is approximately 0.0061: about 4.04 million of the 660.60 million exhaustively fitted structures had an equal or lower AIC.

The structures selected by the exhaustive and sequential searches are shown in Table 1.

The best exhaustive candidate was the Chimera matrix 386,126,232, with 28 fitted parameters, a log-likelihood of 316.5333, and an AIC of −577.0667. Applying the Dißmann algorithm to the same pseudo-observations and copula family set gave an AIC of −560.7392. Thus, in this application, searching the complete eight-dimensional matrix collection identified a model with an AIC 16.33 units lower than the sequentially selected model. The fitted pair copulas of the exhaustive model are reported in Table 2.

Table 1
Regular-vine matrices selected for the ship-wake data by the exhaustive Trasgu search and Dißmann's sequential algorithm.

Trasgu (Chimera 386126232)

$$\begin{pmatrix} 6 & 2 & 4 & 6 & 6 & 6 & 3 & 3 \\ 4 & 6 & 6 & 1 & 4 & 3 & 6 & 0 \\ 8 & 1 & 1 & 4 & 3 & 4 & 0 & 0 \\ 1 & 4 & 2 & 3 & 1 & 0 & 0 & 0 \\ 2 & 3 & 3 & 2 & 0 & 0 & 0 & 0 \\ 3 & 8 & 8 & 0 & 0 & 0 & 0 & 0 \\ 7 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Dißmann

$$\begin{pmatrix} 5 & 1 & 6 & 6 & 4 & 4 & 8 & 8 \\ 6 & 6 & 1 & 4 & 8 & 8 & 4 & 0 \\ 1 & 4 & 4 & 8 & 7 & 7 & 0 & 0 \\ 4 & 5 & 8 & 7 & 6 & 0 & 0 & 0 \\ 2 & 8 & 7 & 1 & 0 & 0 & 0 & 0 \\ 8 & 7 & 5 & 0 & 0 & 0 & 0 & 0 \\ 7 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Table 2
Pair-copula specification of the best exhaustive ship-wake model (Chimera matrix 386126232). Parameters and Kendall's τ are rounded to two decimal places; all variables are continuous.

Tree	Edge	Pair	Conditioning set	Family	Rotation	θ	τ
1	1	5, 6	–	Gaussian	0	0.67	0.46
1	2	7, 2	–	Gaussian	0	0.06	0.04
1	3	8, 4	–	Frank	0	0.67	0.07
1	4	2, 6	–	Clayton	0	0.86	0.30
1	5	1, 6	–	Joe	180	2.06	0.37
1	6	4, 6	–	Gaussian	0	0.52	0.34
1	7	6, 3	–	Joe	0	1.11	0.06
2	1	5, 4	6	Joe	90	1.06	−0.04
2	2	7, 6	2	Joe	270	1.25	−0.13
2	3	8, 6	4	Joe	90	1.15	−0.08
2	4	2, 1	6	Gaussian	0	0.59	0.40
2	5	1, 4	6	Gaussian	0	−0.73	−0.52
2	6	4, 3	6	Joe	180	1.26	0.13
3	1	5, 8	4, 6	Clayton	270	0.05	−0.02
3	2	7, 1	6, 2	Frank	0	1.27	0.14
3	3	8, 1	6, 4	Joe	180	1.31	0.15
3	4	2, 4	1, 6	Frank	0	−3.15	−0.32
3	5	1, 3	4, 6	Joe	180	1.07	0.04
4	1	5, 1	8, 4, 6	Clayton	0	0.26	0.11
4	2	7, 4	1, 6, 2	Joe	90	1.06	−0.03
4	3	8, 2	1, 6, 4	Clayton	180	0.14	0.07
4	4	2, 3	4, 1, 6	Joe	180	1.11	0.06
5	1	5, 2	1, 8, 4, 6	Clayton	0	0.22	0.10
5	2	7, 3	4, 1, 6, 2	Gumbel	90	1.10	−0.09
5	3	8, 3	2, 1, 6, 4	Joe	270	1.08	−0.04
6	1	5, 3	2, 1, 8, 4, 6	Clayton	180	0.29	0.13
6	2	7, 8	3, 4, 1, 6, 2	Joe	270	1.10	−0.05
7	1	5, 7	3, 2, 1, 8, 4, 6	Clayton	180	0.21	0.10

4. Impact

Trasgu reduces the dependence on specialist programming support when preparing and repeating exhaustive fitting campaigns. Its documented configuration and packaged workflow allow researchers with suitable resources to reuse the same implementation while adapting the data, fitting controls and execution profile to their analysis.

Exhaustive fitting provides a reference for evaluating sequential and heuristic structure-selection algorithms. Simulation studies can compare the AIC obtained by a sequential or heuristic algorithm with the lowest AIC obtained across all Chimera structures under the same candidate families, fitting settings and estimation procedure. They can also assess how the results change with the dimension, number of observations, dependence strength or permitted copula families. Exhaustive results can also reveal the multiplicity of similarly scoring structures, their stability across data sets and the effects of restricting the candidate space, supporting a better understanding of vine copula models and their selection procedures.

Trasgu also extends the scale at which Chimera can be investigated. Chimera collections are currently available for up to eight variables; a complete nine-dimensional collection would contain approximately 380.5 billion matrices. The execution and matrix-access infrastructure provided by Trasgu makes generating this collection and exploring it systematically a concrete next research step, although its construction, representation and distribution remain future work.

5. Conclusions

Trasgu provides a lightweight and reproducible execution layer for exhaustive fitting of precomputed Chimera vine structures. Deterministic chunking, parallel execution within chunks, persistent intermediate results and the supplied Snakemake workflow simplify the preparation and execution of searches involving millions of candidate structures. The synthetic and ship-wake experiments demonstrate both the scientific value of exhaustive searches as references for sequential methods and the practical value of resuming an interrupted campaign without repeating completed work. The same workflow can be adapted to local computers and different HPC schedulers through execution profiles. The next step will be to generate and provide access to the nine-dimensional Chimera collection and to evaluate its exhaustive fitting with Trasgu. Future work will also evaluate genuinely distributed cloud execution, including Kubernetes-based deployments.

CRediT authorship contribution statement

Valvanuz Fernández-Quiruelas: Writing – review & editing, Writing – original draft, Visualization, Software, Resources, Methodology, Formal analysis, Conceptualization. **Patricia Mares-Nasarre:** Writing – review & editing, Visualization, Validation. **Cornelis Marcel Pieter 't Hart:** Writing – review & editing, Software. **Oswaldo Morales-Nápoles:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Conceptualization.

Current executable software version

The archived Trasgu v1.0.0 release is described in [19] and is available from PyPI at https://pypi.org/project/trasgu/1.0.0/.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The first author acknowledges the support of the project PID2023-150689OB-I00. The project was funded by MCIN/AEI/10.13039/501100011033 and co-funded by the European Union (ERDF).

Appendix A. Command-line tools

Table A.3 summarizes the command-line tools provided by Trasgu, grouped according to their main function.

Table A.3
Command-line tools provided by Trasgu.

Function	Commands
Preparation and planning	`trasgu_examples`, `trasgu_count_chunks`, `trasgu_time_fit`
Execution and monitoring	`trasgu_run`, `trasgu_monitor`, `trasgu_fit_chunk`
Results	`trasgu_combine`, `trasgu_best_fits`
Chimera matrices	`trasgu_get_matrix`, `trasgu_find_matrix`, `trasgu_fit_given_matrix`, `trasgu_download_zarr`

Data availability

All results, figures and tables reported in this article are fully reproducible. The complete software source code, benchmark protocols and correctness validation suites are publicly available in the Trasgu GitHub repository (https://github.com/SantanderMetGroup/trasgu). All configuration files, workflow snapshots, synthetic datasets and derived experimental artifacts are permanently archived in the accompanying Zenodo data record [20].

Specifically:

- **Benchmarks and validation:** The performance benchmarks in Section 1 and the 96,000-fit correctness validation suites in Section 2 can be reproduced end-to-end using the scripts and synthetic datasets included in the repository.
- **Seven-dimensional experiment:** All 100 synthetic datasets, workflow definitions and chunk outputs are archived on Zenodo, allowing full end-to-end re-execution and immediate reproduction of the AIC comparison in Fig. 5.
- **Ship-wake application:** The source observations are third-party data that can be requested from the original authors [21]. However, all analysis results are fully reproducible without requiring access to the restricted raw data: the processed AIC distribution across all 660,602,880 structures (`processed_aic_cdf.npz`) and the optimal model parameters (`best_fits.txt`) are deposited on Zenodo, allowing readers to reproduce Fig. 6 and verify Tables 1 and 2 with a single provided script. Readers with authorized access to the source CSV can also repeat the full HPC workflow using the provided workflow templates.

References

[1] Joe H. Families of m-variate distributions with given margins and $m(m-1)/2$ bivariate dependence parameters. In: Rüschendorf L, Schweizer B, Taylor MD, editors. Distributions with fixed marginals and related topics, lecture notes–monograph series, vol. 28. Institute of Mathematical Statistics; 1996. p. 120–41. https://doi.org/10.1214/lnms/1215452614

[2] Bedford T, Cooke RM. Probability density decomposition for conditionally dependent random variables modeled by vines. Ann Math Artif Intell 2001;32:245–68. https://doi.org/10.1023/A:1016725902970

[3] Bedford T, Cooke RM. Vines: a new graphical model for dependent random variables. Ann Stat 2002;30(4):1031–68. https://doi.org/10.1214/aos/1031689016

[4] Aas K, Czado C, Frigessi A, Bakken H. Pair-copula constructions of multiple dependence. Insur: Math. Econ. 2009;44(2):182–98. https://doi.org/10.1016/j.insmatheco.2007.02.001

[5] Joe H. Dependence modeling with copulas. Boca Raton: Chapman and Hall/CRC; 2014. https://doi.org/10.1201/b17116

[6] Czado C. Analyzing dependent data with vine copulas: a practical guide with R. In: Lecture notes in statistics, vol. 222. Cham: Springer; 2019. https://doi.org/10.1007/978-3-030-13785-4

[7] Dißmann J, Brechmann EC, Czado C, Kurowicka D. Selecting and estimating regular vine copulae and application to financial returns. Comput Stat Data Anal 2013;59:52–69. https://doi.org/10.1016/j.csda.2012.08.010

[8] Vuong QH. Likelihood ratio tests for model selection and non-nested hypotheses. Econometrica 1989;57(2):307–33. https://doi.org/10.2307/1912557

[9] Morales-Nápoles O, Rajabi-Bahaabadi M, Torres-Alves GA, 't Hart CMP. Chimera: an atlas of regular vines on up to 8 nodes. Sci Data 2023;10:337. https://doi.org/10.1038/s41597-023-02252-6

[10] 't Hart CMP, Morales-Nápoles O, Rajabi M, Torres Alves GA. Chimera – regular vines Python (4 to 8 nodes), 4TU.ResearchData, version 1; 2023. https://doi.org/10.4121/20655120.v1

[11] Miles A, Kirkham J, Durant M, Bourbeau J, Onalan T, Hamman J, et al. Zarr. Zenodo 2020. https://doi.org/10.5281/zenodo.3773450

[12] Morales-Nápoles O. Recent experience with the use of chimera for vine copula modelling and future challenges. In: Matos JC, Lourenço PB, Beer M, Oliveira D, Patelli E, Sousa HS, Silva RA, Santamaria M, editors. Proceedings of the European safety and reliability conference (ESREL 2026). Singapore: Research Publishing; 2026. p. 939–46. https://doi.org/10.3850/ESREL2026061419_esrel26-p26138-cd

[13] Morales-Nápoles O, 't Hart CMP, Torres-Alves GA, Perrone E, Ragno E, Mares-Nasarre P. Benchmarking maximum weight spanning tree selection for simplified vine copulas: simulation and real-data studies. manuscript under preparation 2026.

[14] Köster J, Rahmann S. Snakemake—a scalable bioinformatics workflow engine. Bioinformatics 2012;28(19):2520–2. https://doi.org/10.1093/bioinformatics/bts480

[15] IFCA Advanced Computing and e-Science Group. High performance computing: Altamira supercomputing node at university of Cantabria; 2026. https://confluence.ifca.es/spaces/IC/pages/66176/High+Performance+Computing.

[16] Nagler T, Bumann V, Vatter T. pyvinecopulib: a Python interface to vinecopulib; 2023. https://github.com/vinecopulib/pyvinecopulib.

[17] Schepsmeier U, Stoeber J, Brechmann EC, Graeler B, Nagler T, Erhardt T. VineCopula: statistical inference of vine copulas, R package; 2018. https://cran.r-project.org/package=VineCopula.

[18] Cooke RM, Bedford T. Identifiability and the simplifying assumption. In: Nagler T, Kurowicka D, Cooke RM, Joe H, editors. Statistical dependence modeling. Springer; 2026. p. 109–29. https://doi.org/10.1007/978-3-032-14252-8_6

[19] Fernández-Quiruelas V. Trasgu (version v1.0.0). Computer software. Zenodo; 2026. https://doi.org/10.5281/zenodo.23097343

[20] Fernández-Quiruelas V, Mares-Nasarre P, 't Hart M, Morales-Nápoles O. Experimental data for the trasgu article (version 2.0.0) [dataset]. Zenodo 2026. https://doi.org/10.5281/zenodo.23098246

[21] Muscalus AC, Haas KA, Webster DR. Observations of primary ship waves at the margins of a confined tidal river. J Waterw Port Coast Ocean Eng 2024;150(5):04024009. https://doi.org/10.1061/JWPED5.WWENG-2062