Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

evomap: A Toolbox for Dynamic Mapping in Python

Maximilian Matthe
Indiana University

Abstract

This paper presents **evomap**, a Python package for dynamic mapping. Researchers across disciplines use mapping methods to visualize relationships among objects as spatial representations, or "maps". Most existing statistical software supports only static mapping, which captures objects' relationships at a single point in time and lack tools for analyzing how these relationships evolve over time. **evomap** fills this gap by implementing the dynamic mapping framework EvoMap, initially proposed by Matthe, Ringel, and Skiera (2023), which adapts traditional mapping methods like Multidimensional Scaling for dynamic analysis. The package also offers tools for data preprocessing, exploration, and result evaluation, making it a comprehensive toolkit for dynamic mapping applications across disciplines. This paper details the foundations of static and dynamic mapping, describes **evomap**'s architecture and features, and demonstrates its application with an extensive example.

Keywords: dynamic mapping, multidimensional scaling, proximity analysis, open-source software, Python.

1. Introduction

Mapping, also termed "scaling", encompasses a range of statistical methods that analyze data consisting of pairwise relationships through spatial representations, often referred to as "maps". These methods project pairwise relationships onto configurations where objects are positioned as points within a lower-dimensional space. The resulting maps facilitate exploring the data structure, reveal hidden patterns, and help identify key dimensions of difference.

Prominent techniques such as Multidimensional Scaling (MDS, Carroll and Arabie (1998)), Sammon Mapping (Sammon (1969)), or t-distributed Stochastic Neighbor Embedding (t-SNE, van der Maaten and Hinton (2008)) are extensively used across disciplines including statistics (Saeed, Nam, Al-Naffouri, and Alouini 2019), marketing (DeSarbo, Manrai, and Manrai 1994), psychology (Goodwill and Meloy 2019), psychometrics (Hebart, Zheng, Pereira,

and Baker 2020), ecology (Kenkel and Orlóci 1986), network visualization (Mane and Börner 2004), and political science (Jacoby and Armstrong 2014), where they help understanding market competition, social network structures, ecological interactions, cognitive processes, latent psychological traits, or political landscapes.

Mapping methods traditionally generate static maps based on data that captures pairwise relationships among objects at a specific point in time. However, these relationships often change over time due to evolving consumer perceptions, social interactions, or political stances. When longitudinal data on such evolving relationships is available, analysts can gain deeper insights by representing these changes in dynamic maps. Although dynamic mapping introduces additional methodological challenges, EvoMap, as recently proposed by (Matthe *et al.* 2023), addresses them and provides a flexible framework for adapting static mapping methods, like MDS, Sammon Mapping or t-SNE, to accomodate longitudinal data.

Thus far, however, statistical software development has focused almost entirely on static mapping. In Python, the **scikit-learn** package offers implementations of static mapping methods such as MDS or t-SNE (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, and Dubourg 2011). Individual contributions exist that implement specific methods, such as **sammon-mapping** (Perera 2023), or **umap-learn** (McInnes, Healy, and Melville 2018). In R, the **stats** package provides a function for Classical Scaling, and the **MASS** package provides functions for Sammon Mapping and non-metric MDS (Venables and Ripley 2002). The most comprehensive R package for MDS is **smacof** (De Leeuw and Mair 2009; Mair, Groenen, and de Leeuw 2022), which implements various MDS variants and extensions. Other static mapping methods can be found in **Rtsne** (Krijthe 2015), which implements t-SNE, or **vegan** (Oksanen, Simpson, Blanchet, Kindt, Legendre, Minchin, O'Hara, Solymos, Henry, Stevens, Szoecs, Wagner, Barbour, Bedward, Bolker, Borcard, Carvalho, Chirico, Caceres, Durand, Beatriz, Evangelista, FitzJohn, Friendly, Furneaux, Hannigan, Hill, Lahti, McGlinn, Ouellette, Cunha, Smith, Stier, Braak, and Weedon 2022), **ecodist** (Goslee and Urban 2007), or **SensoMineR** (Le and Husson 2008), which implement variants of non-metric MDS.

These existing packages provide versatile toolsets for static mapping, but their functionality for dynamic mapping is limited. Their functions typically process only a single relationship matrix at a time. When analyzing a temporal sequence of matrices, such independent mapping often creates incomparable solutions that are challenging to compare over time (Matthe *et al.* 2023). Although the **smacof** package supports three-way MDS, which allows to process multiple relationship matrices simultaneously, it is designed to analyze differences across individuals rather than changes over time. Thus, there is a clear gap in software tools dedicated to creating, exploring and evaluating dynamic maps.

This paper presents the Python package **evomap**, designed to fill this gap. Developed as a software companion to Matthe *et al.* (2023), the **evomap** package provides a comprehensive toolbox for dynamic mapping. It offers a flexible implementation of the homonymous *EvoMap* framework compatible with various extant static mapping methods. The package has a highly modular design, facilitating the addition of new mapping methods. Further, it adopts the popular **scikit-learn** syntax, ensuring ease of use, and includes comprehensive modules for evaluating and exploring dynamic mapping results.

This article first provides an overview of static and dynamic mapping, essential for understanding **evomap**'s functionality. It then details the package's design and features, demonstrates its application through an example, and explores advanced considerations for practical use.

The paper concludes with a discussion of potential directions for future development.

2. Background on Mapping

2.1. Static Mapping

Static mapping positions a set of $n \in \mathbb{N}$ objects in a lower-dimensional space such that the distances among them closely reflect their relationships. Typically, the input data consists of a square $n \times n$ matrix of pairwise relationships, usually measured as pairwise (dis-)similarities [1]. The outcome is an n-point configuration in $d \in \mathbb{N}$ dimensions, often visualized as a two-dimensional 'map' (Borg and Groenen 2005). To enhance interpretability, one can augment the maps by incorporating additional features of the analyzed objects, for instance, via property fitting (DeSarbo and Hoffman 1987) or by binding them to visual elements like the size or color of each point (Ringel and Skiera 2016).

Mapping originated in Psychometrics with Torgerson's Classical Scaling (Torgerson 1958). Since then, research from different fields has developed several variants. For instance, Shepard (1962a,b) introduced monotonic transformations of the input dissimilarities, leading to non-metric MDS (Kruskal 1964a,b). Other extensions process measurements from multiple subjects (Carroll and Chang 1970) or link the input dissimilarities to map distances using non-linear functions (Sammon 1969). More recently, other fields, like computer science have contributed methodological innovations, such as t-SNE (van der Maaten and Hinton 2008).

Commonly, these methods fit point coordinates $\hat{X} \in \mathbb{R}^{n \times d}$ of n objects to given data by minimizing the discrepancy between map distances and input dissimilarities. For MDS, this discrepancy is measured through the Stress function:

$$Stress(X) = \sqrt{\frac{\sum_{i,j}(d(x_i, x_j) - \hat{d}_{ij})^2}{\sum_{i,j} d(x_i, x_j)^2}}, \tag{1}$$

where $d(x_i, x_j)$ and $\hat{d}_{i,j}$ denote the map distances and input dissimilarities among objects i and j, respectively (or, depending on the MDS variant, a transformation thereof).

$$\hat{X} = \underset{X \in \mathbb{R}^{n \times d}}{\arg\min}\, \text{Stress}(X), \tag{2}$$

The quality of $\hat{X}$ can be evaluated in multiple ways, including the Stress value, different agreement metrics like the Hit-Rate of K-nearest neighbor recovery (Chen and Buja 2009), or through visual comparisons, like the Shepard Diagram.

Figure 1 illustrates non-metric MDS applied to a sample of nine technology firms. The map is generated based on the textual similarity of the firms' product descriptions in 1998, using data from the TNIC dataset (Hoberg and Phillips 2016). The map (left) reveals that, within the nine depicted firms, there are four groups of similar firms based on their main products sold in 1998: i) hardware-focused firms, such as Apple, Western Digital, and Micron Technologies; ii)

[1] Although input data can vary, this discussion focuses on dissimilarities, the most prevalent type, for simplicity.

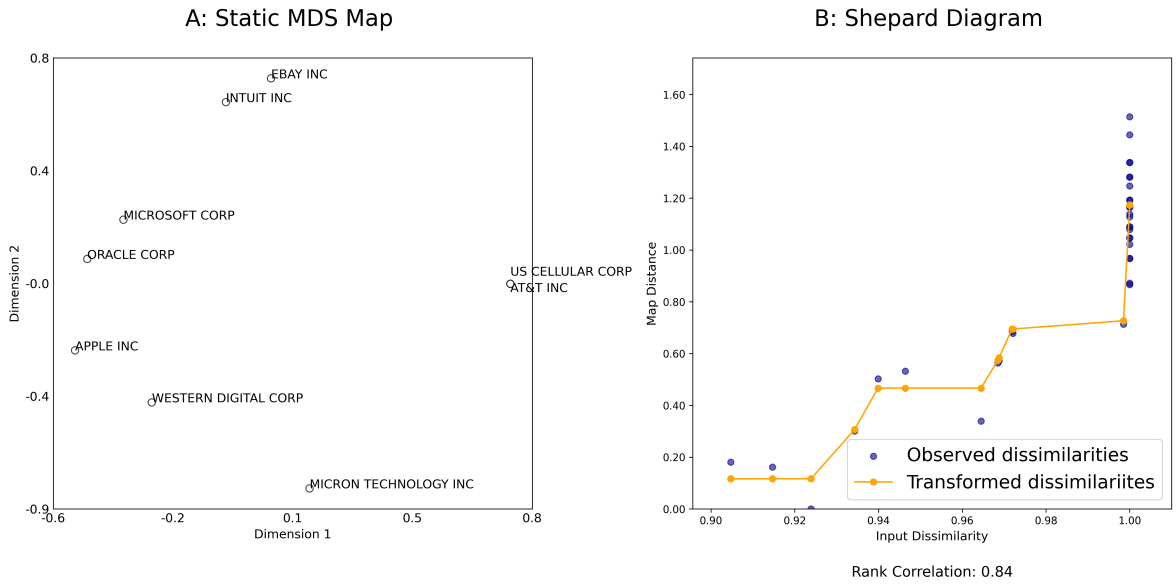

Figure 1: Illustration of Static Mapping

platform-focused firms, like Microsoft and Oracle, iii) software-focused firms, such as eBay and Intuit, and iv) telecommunication service providers, like AT&T and US Cellular. Among these pairs, the telecommunication firms appear more distinct compared to the remaining groups whose products commonly relate to computers. Within the three computer-related groups, Microsoft and Oracle, known for platforms like the Windows Operation System or Oracle's Database System, appear between software and hardware providers. Among the hardware providers, Micron Technologies, primarily known as a semiconductor firm, is differentiated from Western Digital, known for storage solutions, and Apple, known for personal computers like the iMac.

The Shepard Diagram (right) confirms the map's good solution quality. The transformed dissimilarities fit the observed input dissimilarities well, and the rank correlation between the observed input dissimilarities and the map distances remains high at .84.

As these firms change their product offerings, their market positions change. Thus, a map of the same firms generated at different points in time would likely look different. If longitudinal measurements are available, the analyst could track the evolution of these maps, gaining insights into the timing, extend, and duration of such changes. These are precisely the insights dynamic mapping seeks to provide.

2.2. Dynamic Mapping

Dynamic mapping extends static mapping by revealing how relationships among n objects change over time. In dynamic mapping, the input data consist of a sequence of relationship matrices $(D_t)_{t=1,\ldots,T}$ rather than a single matrix. Applying static mapping methods to each time-point matrix D_t independently introduces several challenges. The estimated point configurations $(\hat{X}_t)_{t=1,\ldots,T}$, where $X_t \in \mathbb{R}^{n\times d}$, do often not align over time, preventing the analyst from tracking objects' movement. Further, results can be inconsistent due to local optima

and are highly sensitive to variations in the data, including noise.

EvoMap, as proposed by Matthe *et al.* (2023), addresses these issues by incorporating static mapping methods into a dynamic optimization framework. *EvoMap* jointly fits a sequence of point configurations $(\hat{X}_t)_{t=1,\dots,T}$ to a sequence of dissimilarity matrices $(D_t)_{t=1,\dots,T}$, linking them over time through a regularization component. Thereby, it not only aligns subsequent maps but also smooths the trajectories of objects on the map. As a result, *EvoMap* produces coherent spatial representations of the evolving relationships among the objects under analysis ("dynamic maps").

Specifically, *EvoMap*'s optimization problem is defined as:

$$(\hat{X}_t)_{t=1,\dots,T} = \underset{X_1,\dots,X_T \in \mathbb{R}^{n\times d}}{\arg\min} C_{total}(X_1,\dots,X_T), \tag{3}$$

with the total cost function:

$$C_{total}(X_1,\dots,X_T) = \sum_{t=1}^{T} C_{static}(X_t) + \alpha \cdot C_{temporal}(X_1,\dots,X_T), \tag{4}$$

Here, C_{static} represents the static component of EvoMap's cost function, which equals the sum of the cost function C_{static} of the selected static mapping method evaluated at each period $t \in \{1,\dots,T\}$. When paired with MDS, for instance, C_{static} would equal the Stress function in equation 1. $C_{temporal}$, weighted by the hyperparameter $\alpha \in \mathbb{R}$, represents the temporal component of the cost function, which aligns the maps and smooths objects' movements over time. The temporal cost component, $C_{temporal}$, is further defined as:

$$C_{temporal}(X_1,\dots,X_T) = \sum_{i=1}^{n} f_w(i) \sum_{k=1}^{p} \sum_{t=k=1}^{T} \mathbb{1}_{\{i\in I\}} \left\| \nabla^k x_{i,t} \right\|^2, \tag{5}$$

where

$\mathbb{1}_{\{i\in I_{t,k}\}} : I \to \{0,1\}$ denotes an indicator function equal to one if firm i is present in time t and the k preceding periods: $\mathbb{1}_{\{i\in I_{t,k}\}} := \prod_{l=0}^{k} \mathbb{1}_{\{i\in I_{t-l}\}}$,

$f_w : I \to \mathbb{R}^+$ is a positive weight function defined over the set of firms I,

$p \in \mathbb{N}$ is a second hyperparameter of positive integer values that controls the degree of smoothing, and

$\nabla^k x_{i,t} \in \mathbb{R}^d$ is the k-th (backward) difference of firm i's map position at time t.

We refer to Matthe *et al.* (2023) for more details and intuition on *EvoMap*'s cost function.

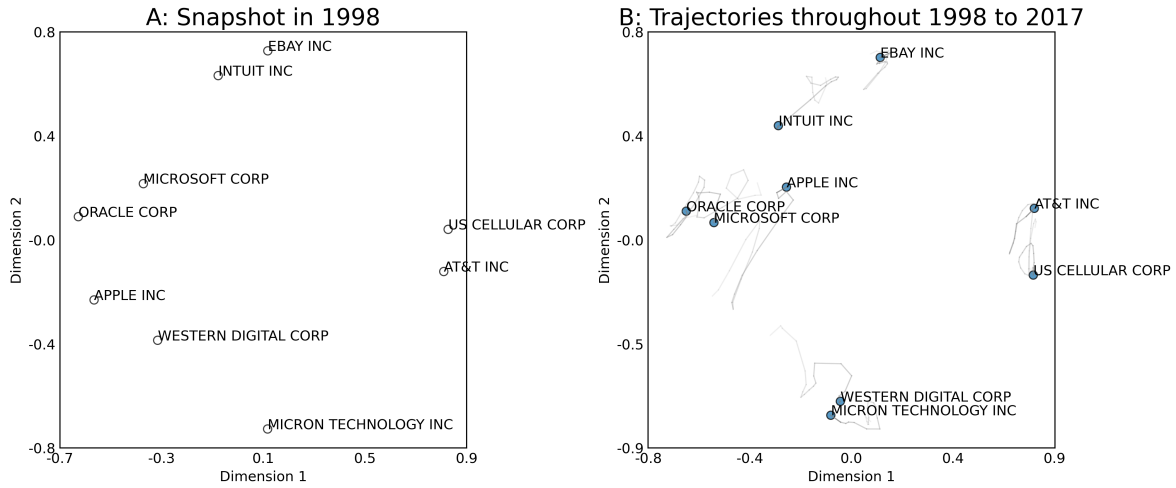

Figure 2: Illustration of Dynamic Mapping

Figure 2 demonstrates dynamic mapping for nine technology firms over 20 years. Using *EvoMap*, we generated a sequence of 20 snapshots; the left graph displays the first snapshot from 1998, while the right graph depicts the firms' evolving positions over the subsequent two decades. The dynamic map on the right highlights key shifts in the firms' positions: Apple and Western Digital diverge as Apple ventures into the software space, drawing it closer to firms like Intuit. Meanwhile, Microsoft and Oracle converge, and the positions of telecommunication firms remain relatively static. The remainder of this paper details how to create such dynamic maps using the **evomap** package.

3. The evomap Package

3.1. Installation

evomap can be installed directly from the Python Package Index (PyPI) by running:

```
> pip install evomap
```

The installation requires Python version 3.9 or newer. At the time of writing the latest release of **evomap** is version 0.4.4. Updates are actively developed on GitHub, available at https://github.com/mpmatthe/evomap and subsequently released to PyPI.

3.2. Key Dependencies

evomap relies on a limited number of other packages. It uses **numpy** and **scipy** for numerical computations, and **matplotlib** and **seaborn** for plotting. Additionally, **evomap** uses individual routines from **scikit-learn** and **statsmodels**, and employs the just-in-time C-compiler **numba** to optimize runtime.

3.3. Package Overview

evomap is structured into six core modules[2], designed to streamline the analysis workflow:

1. `datasets`: Offers example datasets, enabling users to familiarize themselves with the package's capabilities.
2. `preprocessing`: Transforms diverse input data into a format compatible with EvoMap.
3. `mapping`: Contains EvoMap implementations for various mapping methods.
4. `printer`: Provides tools for visual exploration of both static and dynamic maps.
5. `transform`: Includes options for adjusting maps (e.g., rotations and reflections that can enhance interpretability).
6. `metrics`: Features a set of evaluation metrics and goodness-of-fit statistics to evaluate mapping results.

Table 1 presents an overview of these modules, with detailed descriptions and a complete API reference available at ReadTheDocs.

#	Name	Purpose	Exemplary Functions or Classes
(1)	evomap.datasets	Load exemplary datasets	`load_tnic_snapshot()`
(2)	evomap.preprocessing	Prepare data for mapping	`coocc2sim()`, `edgelist2matrix()`
(3)	evomap.mapping	Apply EvoMap to data	`MDS()`, `EvoMDS()`, `TSNE()`, `EvoTSNE()`
(4)	evomap.printer	Visualize mapping results	`draw_map()`, `draw_dynamic_map()`
(5)	evomap.transform	Adjust mapping results for easier interpretation	`align_map()`
(6)	evomap.metrics	Assess mapping quality	`misalign_score()`, `hitrate_score()`, `persistence_score()`

Table 1: Overview of Modules

3.4. Basic Usage

The primary use case for **evomap** is transforming longitudinal relationship data into dynamic low-dimensional spatial representations. The process is straightforward and involves the following three steps:

1. **Import the EvoMap class** for the desired mapping method

[2]For clarity, the unified term 'module' encompasses submodules, modules, and subpackages.

2. **Initialize EvoMap** with appropriate parameters
3. **Fit EvoMap** to your data using `fit_transform()`

For example, consider longitudinal data represented by a sequence of 20 dissimilarity matrices for 9 firms, each stored as a **numpy** array of shape `(9,9)`.

```
>>> print(len(D_t))       # Output: 20
>>> print(D_t[0].shape)   # Output: (9,9)
```

To apply non-metric MDS, instantiate `EvoMDS()`, specify the `mds_type` as `'ordinal'`, set its hyperparameters α and p (whose role we detail later), and provide initial starting positions `init`. Then, fit EvoMap to your data:

```
>>> from evomap.mapping import EvoMDS
>>> evomds = EvoMDS(
...     alpha=.2,
...     p=2,
...     mds_type='ordinal',
...     init=cmds_t)
>>> X_t = evomds.fit_transform(D_t)
```

The result, `X_t`, is a sequence of `numpy` arrays, positioning the firms on two dimensions over 20 years.

```
>>> print(len(X_t))   # Output: 20
>>> print(X_t[0].shape)   # Output: (9,2)
```

To explore the result, we can visualize the point coordinates stored in `X_t` with `draw_dynamic_map()`, providing labels, and optionally displaying movement arrows and axes. Figure 3 showcases the resultant evolution of the nine firms' positions from 1998 to 2017.

```
>>> from evomap.printer import draw_dynamic_map
>>> draw_dynamic_map(
...     X_t,
...     label=labels,
...     show_arrows=True,
...     show_axes=True)
```

The **evomap** package encompasses a full toolbox of functionalities beyond simply generating dynamic maps. It equips users with a range of tools for each step of the data analysis process, from initial data preprocessing and formatting to exploring and assessing mapping results. The following section briefly illustrates the overall package design and implementation, before the paper presents a comprehensive usage example.

3.5. Package Design and Implementation

The **evomap** package is centered around its `mapping` module. This module is the heart of **evomap**, implementing the *EvoMap* framework for various mapping methods along with

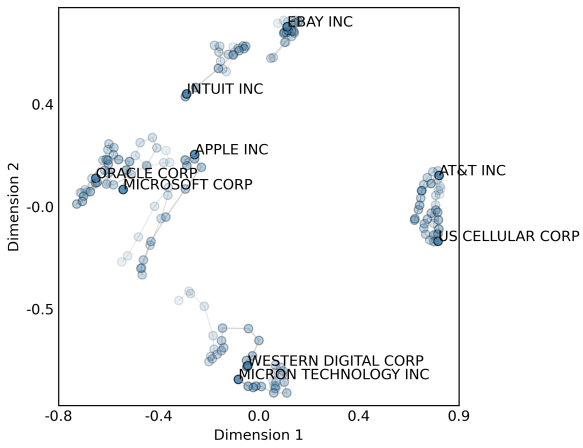

Figure 3: Dynamic Map of Nine Technology Firms, 1998 - 2017

their optimization routines. As detailed in section 2.2, *EvoMap*'s cost function contains a static and a temporal component - the former varies by mapping method, while the latter is consistent across all *EvoMap* implementations. Accordingly, the `mapping` module follows a dual structure as well. There, a static implementation of each mapping method provides its static cost component, while a dynamic implementation integrates the method into the *EvoMap* framework. For MDS, for instance, the `mapping` module includes `MDS` (static) and `EvoMDS` (dynamic).

This dual structure enables users to employ mapping methods in both static and dynamic contexts. More importantly, it provides a modular structure than can easily be extended to incorporate new mapping methods. To add a new method, one must only introduce a static class defining the static cost function and a new instance of the EvoMap `core` interface. Figure 4 visualizes the package architecture and highlights this modularity. Common functionalities are provided by dedicated classes, such as `optim` or `core`, and shared across implementations. As a result, extending the framework is easy, and all enhancements to the core framework affect all mapping implementations uniformly.

Currently, **evomap** provides implementations for three mapping methods (MDS, Sammon Mapping, t-SNE), which can serve as blueprints for future extensions. Having presented the general design and structure of the package, the next section illustrates its practical usage.

Note that the results reported in this paper were generated using Python version 3.12.0 and **evomap** version 0.4.4 on macOS 13.5.2. While the **evomap** package is designed to be platform-independent, minor differences in results may occur across operating systems (e.g., due to variations in floating-point precision). However, results on other systems are expected to be very similar, and any differences should not affect the qualitative conclusions drawn from the analyses.

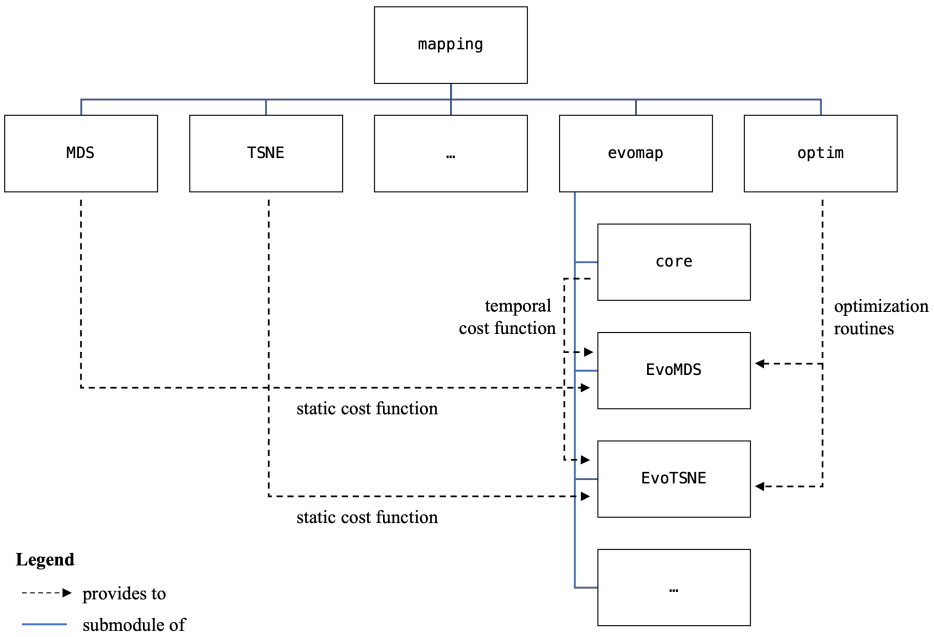

Figure 4: Overview of the Mapping Module

4. Detailed Usage Example

This section demonstrates how to apply the **evomap** package to a subset of the Text-based Network Industries Classification (TNIC) dataset by Hoberg and Phillips (2016). It provides a step-by-step guide, showcasing the tools within **evomap** that facilitate each phase of the analysis.

The TNIC dataset captures firm-firm relationships based on the linguistic similarity of their product descriptions in annual reports. The small subset used for this illustration focuses on nine technology companies: Apple, AT&T, eBay, Intuit, Micron Technology, Microsoft, Oracle, US Cellular, and Western Digital, spanning twenty years from 1998 to 2017. Each data point represents a pair of firms at a given time, including their names, a similarity score, Standard Industrial Classification (SIC) codes, and a size variable derived from their market values. To load this subset, run:

```
>>> from evomap.datasets import load_tnic_sample_tech
>>> data = load_tnic_sample_tech()
```

Table 2 lists the first and last five rows to illustrate the data structure. Each row details the year, firm names (name1, name2), their pairwise similarity score, SIC codes (sic1, sic2), and size variables (size1, size2).

4.1. Preprocessing

The **evomap** package is designed to analyze longitudinal relationship data, requiring input as a sequence of $n \times n$ relationship matrices, where n represents the number of objects.

row	year	name1	name2	score	sic1	sic2	size1	size2
0	1998	APPLE INC	WESTERN DIGITAL CORP	0.0657	36	35	71.79	32.29
1	1998	APPLE INC	MICROSOFT CORP	0.0601	36	73	71.79	517.38
2	1998	APPLE INC	ORACLE CORP	0.0355	36	73	71.79	188.44
3	1998	AT&T INC	US CELLULAR CORP	0.0761	48	48	324.14	57.62
4	1998	EBAY INC	MICROSOFT CORP	0.0281	73	73	98.54	517.38
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
437	2017	ORACLE CORP	MICROSOFT CORP	0.1292	73	73	432.13	728.91
438	2017	ORACLE CORP	INTUIT INC	0.0231	73	73	432.13	187.30
439	2017	US CELLULAR CORP	AT&T INC	0.0184	48	48	56.56	488.57
440	2017	WESTERN DIGITAL CORP	APPLE INC	0.0321	35	36	161.40	888.85
441	2017	WESTERN DIGITAL CORP	MICRON TECHNOLOGY INC	0.0788	35	36	161.40	188.55

Table 2: Overview of the TNIC Sample Data

These matrices should contain non-negative real numbers, typically representing similarities or dissimilarities. Each matrix should be symmetric, normalized (usually between 0 and 1), and free of missing values. The dataset should be balanced, with a consistent set of objects across time. A later section shows how to handle unbalanced data as well.

Many mapping methods, like MDS, require dissimilarities as their input. When the available data does not directly fit this requirement, **evomap** offers preprocessing functions to convert various data types into a suitable format.

For the TNIC example, which is initially provided as a time-indexed edgelist and measures similarities rather than dissimilarities, the first step involves converting the edgelist into a sequence of matrices. This is achieved using the `edgelist2matrices()` function, specifying columns for object identifiers, similarity scores, and time indicators.

```
>>> from evomap.preprocessing import edgelist2matrices
>>> S_t, labels_t = edgelist2matrices(
...     data,
...     score_var='score',
...     id_var_i='name1',
...     id_var_j='name2',
...     time_var='year')
```

The output, `S_t`, now contains the data in the required format for EvoMap: a sequence of

square matrices, while `labels_t` indicates the objects' names along the rows/columns of each matrix.

The next step involves converting the similarity measures in `S_t` into dissimilarities. Here, we use a straightforward transformation, $diss_{i,j} = 1 - sim_{i,j}$, which, alongside other transformations, is available in the `sim2diss()` function. We apply this transformation to each matrix in the sequence, which yields a list of 20 dissimilarity matrices, each of shape $(9, 9)$, ready for dynamic mapping with EvoMap.

```
>>> from evomap.preprocessing import sim2diss
>>> D_t = [sim2diss(S, transformation = 'mirror') for S in S_t]
```

4.2. Dynamic Mapping with EvoMap

Dynamic mapping with EvoMap involves initializing the desired mapping method and executing the `fit_transform()` function. EvoMap then transforms the preprocessed dissimilarity matrices into dynamic maps that represent the evolving relationships among objects over time. For example, to use EvoMDS for Multidimensional Scaling, run:

```
>>> evomds = EvoMDS(
...     alpha=0.2,
...     p=2,
...     mds_type='ordinal',
...     init=cmds_t)
>>> X_t = evomds.fit_transform(D_t)
```

The key parameters to consider when initializing EvoMap include:

- **Starting Positions**, `init`, which can significantly influence the optimization process and final mapping quality.
- **EvoMap's Hyperparameters**, α and p, which regulate the degree of alignment and smoothness.
- **Optimization Arguments**, which impact convergence and computational time. For example, `n_iters` sets the number of iterations for the optimization algorithm.
- **Method-Specific Arguments**, which vary by the mapping method used. For instance, MDS requires the specify the `mds_type` (e.g., ratio, interval, or ordinal), while t-SNE requires to set its `perplexity`.

The following sections outline these choices in more detail.

Starting Positions

To achieve consistent results across different runs, it's important to control the initialization of the optimization process. This can be done in two ways:

1. **Setting a Fixed Seed**, by using `np.random.seed()` with a specific number before instantiating EvoMap.

2. **Setting Starting Positions**, by using the `init` argument to provide predetermined starting positions when instantiating EvoMap.

When setting starting positions, the `init` parameter accepts a list of starting positions matching the desired output shape, in this case, `(9,2)`. A common strategy is to use the output of another (static) mapping method, such as Classical Scaling, for these starting positions (De Leeuw and Mair 2009).

Alternatively, one can also set the `n_inits` argument to a positive integer. Since mapping methods can be sensitive to the choice of starting positions for gradient descent, it's always advisable to compare results across different choices. When `n_inits` is greater than one, the optimization runs `n_inits` times with different random starting positions, reports convergence results for all runs, and stores the best solution (i.e., the one with the lowest cost value).

In our example, we employ a different strategy and use the Classical Scaling solution from the first time period as the starting positions for all periods. Classical Scaling is fully deterministic, does not rely on random seeds, and offers full reproducibility - which makes it well suitable to generate starting positions. Classical Scaling can be applied via `CMDS` in `evomap.mapping`, similar to other mapping methods, by using `fit_transform()`. Using `CMDS`, we generate a list of identical starting positions for each period:

```
>>> from evomap.mapping import CMDS
>>> cmds_t = []
>>> cmds = CMDS().fit_transform(D_t[0])
>>> for t in range(n_periods):
...     cmds_t.append(cmds)
```

EvoMap's Hyperparameters

EvoMap requires the user to set two hyperparameters: α and p. α influences how strongly subsequent point coordinates align with each other, whereas p controls the smoothness of each object's movement path across time. Carefully selecting these hyperparameters is crucial, as they strongly affect EvoMap's output.

Users of the **evomap** package can identify suitable values for α and p in two primary ways:

1. **Visual Inspection**, by exploring EvoMap's output under various hyperparameter combinations.

2. **Quantitative Evaluation**, by using a grid search to systematically test and evaluate a range of hyperparameter combinations.

To demonstrate the influence of these hyperparameters, we examine the effects of different α and p values on EvoMap's output using the TNIC dataset as an example. Figure 5 displays dynamic mapping results generated with different hyperparameter choices, showcasing how each parameter alters the mapping results. The first row varies α (holding p constant at 1), the second row varies p (holding α constant at a medium value).

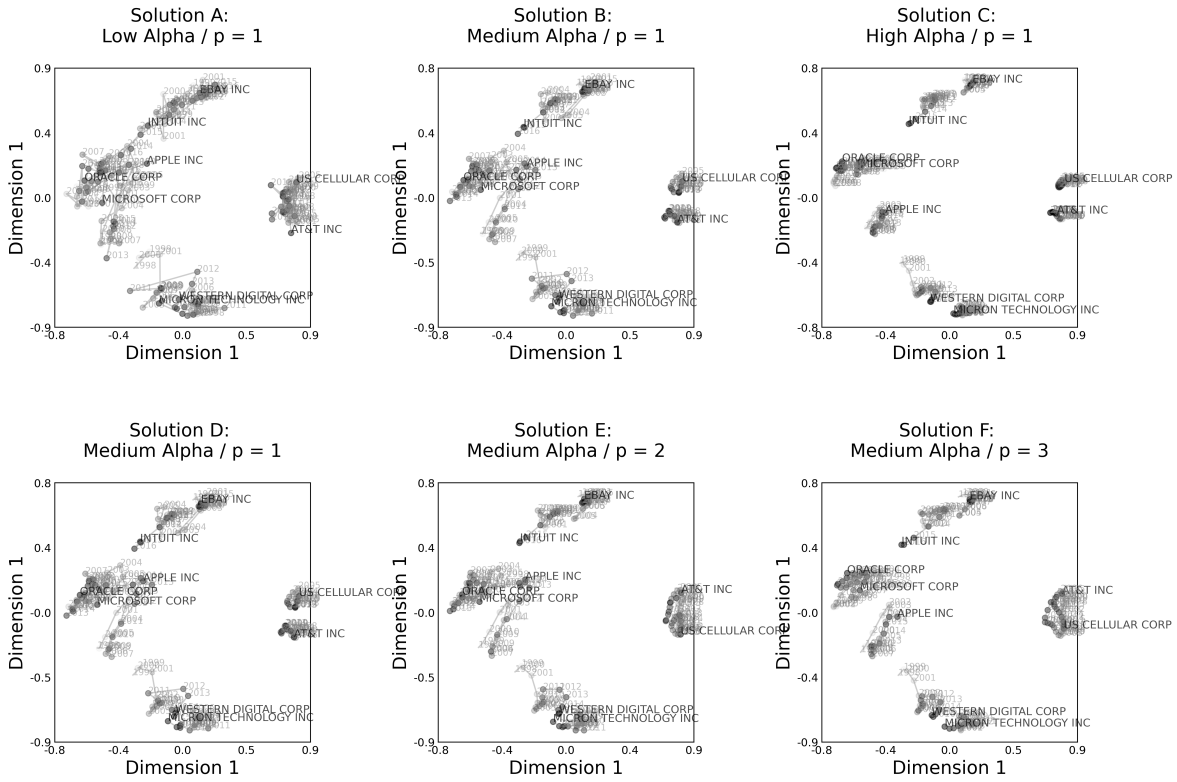

Figure 5: Dynamic Map Under Different Hyperparameter Choices

Figure 5 illustrates that a lower α value tends to produce maps with weaker alignment and more erratic movement paths. As α increases, maps become more aligned, and the movement paths shorten. An excessively high α might lead to nearly static maps, showing only minimal changes over time. Conversely, the impact of p is less about the length of the paths and more about their smoothness; higher p values decrease zigzag movements, leading to smoother trajectories and more uniform distances between subsequent points.

A later section will discuss a more methodological approach, a grid search, for selecting appropriate values for α and p.

Convergence Diagnostics

evomap operates quietly by default, providing no details about the convergence of its gradient-based optimization. However, such information on convergence can be crucial, especially when faced with suboptimal solution quality or unexpected results. To obtain such diagnostics, users can modify the `verbose` parameter to increase output verbosity:

- `verbose=0` (the default) means no output is provided.
- `verbose=1` reveals basic information about the optimization routine, the final cost function value, and convergence status.
- `verbose=2` offers detailed insights into the optimization process, including cost function

values at specific intervals.

Setting `verbose=1`, for instance, indicates the optimization routine (gradient descent with backtracking) and an indication of its successful convergence at iteration 105:

```
>>> EvoMDS(
...   alpha=.2,
...   mds_type='ordinal',
...   init=cmds_t,
...   verbose=1
... ).fit(D_t)
```

```
[EvoMDS] Running Gradient Descent with Backtracking via Halving
[EvoMDS] Iteration 535: gradient norm vanished. Final cost: 3.87
```

Here, we use `fit()`, which runs EvoMap without returning the resultant point coordiantes. Setting `verbose=2` and `n_iter_check=50` evaluates the cost function every 20 iterations and reveals how the cost function values decrease monotonically, until the gradient norm vanishes around iteration 105.

```
>>> EvoMDS(
...     alpha=.2,
...     mds_type='ordinal',
...     init=cmds_t,
...     n_iter_check=50,
...     verbose=2
... ).fit(D_t)
```

```
[EvoMDS] Running Gradient Descent with Backtracking via Halving
[EvoMDS] Iteration 50 -- Cost: 3.90 -- Gradient Norm: 0.0176
[EvoMDS] Iteration 100 -- Cost: 3.88 -- Gradient Norm: 0.0089
[EvoMDS] Iteration 150 -- Cost: 3.88 -- Gradient Norm: 0.0100
[EvoMDS] Iteration 200 -- Cost: 3.88 -- Gradient Norm: 0.0059
[EvoMDS] Iteration 250 -- Cost: 3.88 -- Gradient Norm: 0.0052
[EvoMDS] Iteration 300 -- Cost: 3.88 -- Gradient Norm: 0.0045
[EvoMDS] Iteration 350 -- Cost: 3.88 -- Gradient Norm: 0.0024
[EvoMDS] Iteration 400 -- Cost: 3.87 -- Gradient Norm: 0.0121
[EvoMDS] Iteration 450 -- Cost: 3.87 -- Gradient Norm: 0.0037
[EvoMDS] Iteration 500 -- Cost: 3.87 -- Gradient Norm: 0.0046
[EvoMDS] Iteration 535: gradient norm vanished. Final cost: 3.87
```

Should these diagnostics indicate issues with convergence, several strategies can help improve optimization performance. Specifically, the user can:

- Utilize `evomap.preprocessing` to normalize input data, as some mapping methods struggle with input data measured on unusual or very large scales.
- Increase `n_iter` to extend the number of allowed iterations.
- Adjust `step_size` (by default: 1) to modify convergence speed; decrease if optimization overshoots, or increase it for faster convergence.
- Experiment with multiple initializations through `n_inits` to mitigate the impact of random starting positions.

Once the user is confident about proper convergence of the optimization, one can start exploring the results in more depth and evaluate the solution quality more rigorously.

4.3. Exploration

The **evomap** package provides a range of options to explore mapping results visually through the `printer` module. One can categorize these functions into two groups:

1. **Static Maps**, which visualize individual snapshots at a time
2. **Dynamic Maps**, which visualize the evolution over time

Static Maps

Static snapshots help evaluate the quality of mapping solutions and set reference points for dynamic exploration. The primary function for static visualization is `draw_map()`, which, in its simplest form, requires just the point coordinates $\hat{X}$ to generate a scatter plot. For unidimensional data, it plots a line scale. Figure 6 displays three examples of static snapshots.

```
>>> fig, ax = plt.subplots(1,3)
>>> draw_map(X_t[0], label=labels, ax=ax[0])
>>> draw_map(X_t[10], label=labels, ax=ax[1])
>>> draw_map(X_t[19], label=labels, ax=ax[2])
```

Further customization is possible by binding additional information about each object to the map's aesthetics, such as labels, colors, and sizes, as illustrated in Figure 7.

```
>>> draw_map(X_t[0])
>>> draw_map(X_t[0], label=labels)
>>> draw_map(X_t[0], label=labels, color=sic_codes)
>>> draw_map(X_t[0], label=labels, color=sic_codes, size=sizes)
```

Dynamic Maps

Beyond individual snapshots, **evomap** allows to create dynamic maps, overlaying multiple snapshots to illustrate changes over time. This is accomplished with `draw_dynamic_map()` or `draw_trajectories()`.

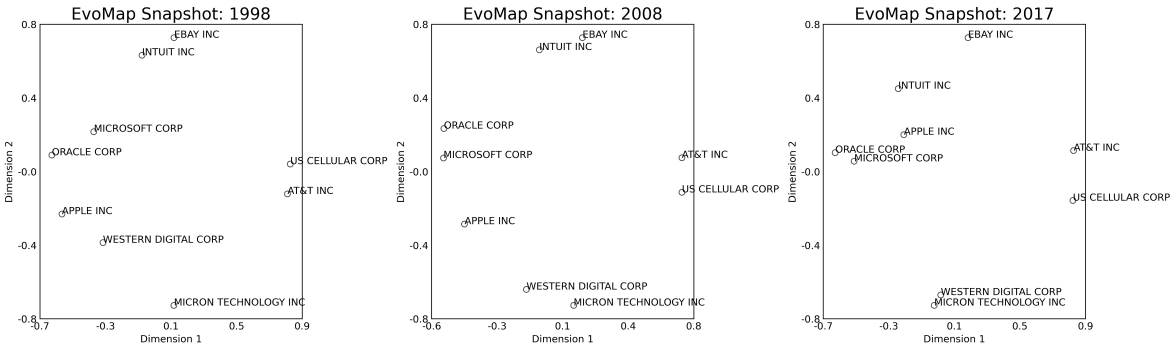

Figure 6: Three Static EvoMap Snapshots

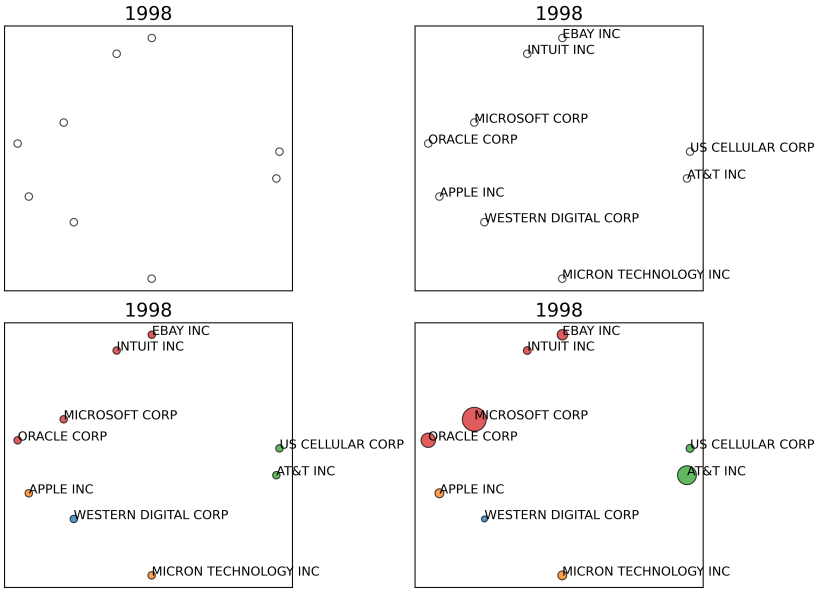

Figure 7: Static Map with Different Aesthetics

Dynamic maps incorporate the same aesthetic customizations as static maps but extend them to reflect changes across time points. Thus, variables like color or size should be provided as lists of arrays, each corresponding to a specific time point.

An example of creating a dynamic map with additional aesthetics is:

```
>>> from evomap.printer import draw_dynamic_map
>>> draw_dynamic_map(X_t,
...     label=labels,
...     color_t=sic_codes_t,
...     size_t=sizes_t,
...     show_arrows=True,
...     show_axes=True)
```

To focus solely on the movement paths, `draw_trajectories()` offers a specialized view high-

lighting the trajectory of each object through time.

```
>>> from evomap.printer import draw_trajectories
>>> draw_trajectories(X_t,
...     labels=labels,
...     period_labels=periods,
...     show_axes=True)
```

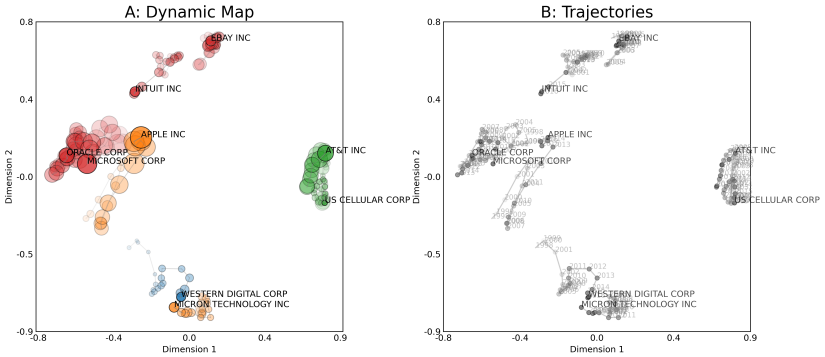

Figure 8: Dynamic Maps for TNIC Sample

4.4. Evaluation

While visual exploration allows to explore the mapping results, it is essential to evalute how well they represent the underlying data. This can be achieved through:

1. **The Cost Function**, which is useful for comparing solution quality at different points in time or comparing different settings within the same method.

2. **Dedicated Metrics**, offered in the `metrics` module, which provide a systematic way to assess mapping quality and compare it across methods.

For instance, in the TNIC dataset example, we assess how EvoMap's alignment affects individual mapping solution quality by comparing the average Stress against a sequence generated independently (i.e., with alpha set to 0). The increase in Stress is negligible (+0.0075), which indicates a minimal decrease in static quality for the benefit of alignment.

```
>>> cmds_indep = []
>>> for t in range(n_periods):
...     cmds_indep.append(CMDS().fit_transform(D_t[t]))
>>> evomds_indep = EvoMDS(
...     alpha=0,
...     init=cmds_indep,
...     mds_type='ordinal'
... ).fit(D_t)
>>> print(evomds_indep.cost_static_avg_.round(4))
>>> print(evomds.cost_static_avg_.round(4))
```

```
0.1875
0.1911
```

Table 3 introduces the additional evaluation metrics available in the `metrics` module. The table lists the metric, its purpose, application scope (single snapshot vs. sequence), and interpretation range. We refer to Matthe *et al.* (2023) for more details on these metrics and their underlying intuition.

Metric	**Intuition**	**Snapshot vs. Sequence**	**Range**
Misalignment	Total length of all movement paths.	Sequence	0 (good) to Inf (poor)
Alignment	Average cosine similarity of subsequent positions.	Sequence	-1 (poor) to 1 (good)
Hit-Rate	Agreement in nearest neighbors between data and the map.	Snapshot	0 (poor) to 1 (good)
Adjusted Hit-Rate	Hit-Rate, adjusted for random agreement.	Snapshot	0 (poor) to 1 (good)
Average Hit-Rate	Average Hit-Rate over multiple periods.	Sequence	0 (poor) to 1 (good)
Average Adjusted Hit-Rate	Average adjusted Hit-Rate over multiple periods.	Sequence	0 (poor) to 1 (good)
Persistence	Correlation of subsequent movement vectors on the map.	Sequence	-1 (poor) to 1 (good)

Table 3: Evaluation Metrics

In the TNIC example, we compute the following metrics: alignment and persistence. We compute each metric for EvoMap's output and two benchmarks: independent application of MDS and ex-post alignment of independent mapping via Procrustes Analysis. Table 4 displays the results. As seen in the Table, EvoMap's solution is better aligned and movement paths are smoother (thus, easier to interpret), while static mapping quality is not reduced.

```
>>> from evomap.metrics import *
>>> misalign_score(X_t).round(4)
>>> persistence_score(X_t).round(4)

0.0468
0.6484
```

Test	Misalignment Score	Persistence Score	Static Cost
Independent MDS	0.9737	-0.5589	0.1875
Independent MDS+ Alignment	0.3074	-0.2718	0.1875
EvoMDS	0.0468	0.6484	0.1911

Table 4: Evaluation Results

5. Advanced Features

To close the presentation of the **evomap** package, this final section showcases two additional considerations that are relevant in many applications: finding good hyperparameter values and dealing with unbalanced data.

5.1. Finding Good Hyperparameter Values

Determining good values for the hyperparameters α and p is critical, yet there's no one-size-fits-all solution. The appropriateness of these values varies significantly across different applications, influenced by factors such as the scale of the static cost function, the number of objects, and the length of the observation period. Consequently, identifying suitable hyperparameter values for EvoMap involves an exploratory process.

To facilitate this process, **evomap** includes functionality for conducting a grid search over a range of hyperparameter values, allowing users to quantitatively assess each combination and identify ranges of promising values.

To run the grid search, the user needs to:

1. Set a hyperparameter grid:

```
>>> param_grid = {
...     'alpha': np.linspace(0, 1.5, 15),
...     'p': [1, 2]}
```

2. Select evaluation metrics:

```
>>> from evomap.metrics import misalign_score, persistence_score, avg_hitrate_score
>>> metrics = [misalign_score, persistence_score, avg_hitrate_score]
>>> metric_labels = ['Misalignment', 'Persistence', 'Hitrate']
```

3. Perform the grid search:

```
>>> model = EvoMDS(init = cmds_t, mds_type = 'ordinal')

>>> df_grid_results = model.grid_search(
...     Xs=D_t,
...     param_grid=param_grid,
...     eval_functions=metrics,
...     eval_labels=metric_labels,
...     kwargs={"input_format" : 'dissimilarity'})
```

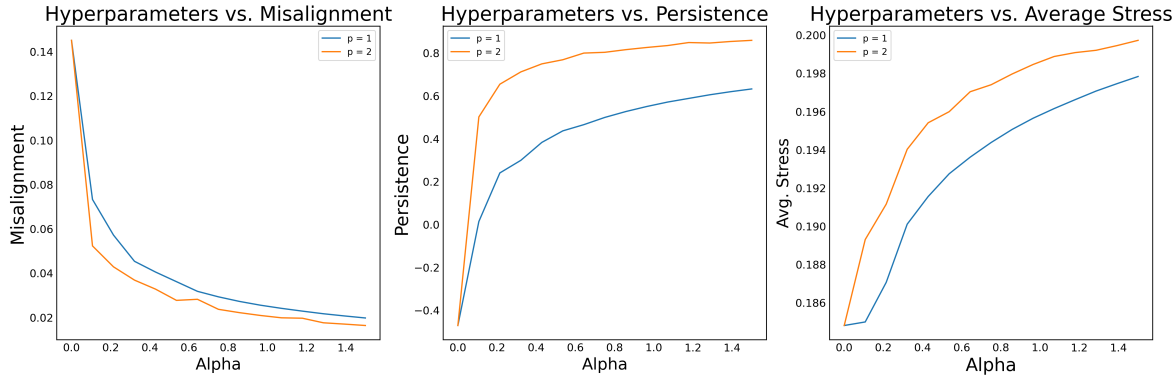

Figure 9: Grid Search Results

The grid search (Figure 9) reveals a sharp decrease in Misalignment scores within the α range of .2 to .4, suggesting this as a viable range for further examination. Within this range, setting $p = 2$ enhances Persistence without significantly compromising the static fit, as indicated by the moderate increase in average Stress.

This exploratory approach allows users to narrow down potential hyperparameter values that balance static and dynamic mapping quality. Once these potential values are identifed, the user can inspect them further using visual exploration.

5.2. Dealing with Unbalanced Data

Dynamic mapping often encounters scenarios where the set of objects changes over time, such as firms entering or exiting a market. The **evomap** package accommodates such unbalanced data through inclusion vectors.

To maintain a consistent data format across time, despite the unbalanced nature of the dataset, **evomap** requires each relationship matrix in the sequence to have the same dimensions. This is achieved by:

1. **Balancing the Data**, by converting it into a balanced sequence of matrices of uniform shape, incorporating all objects observed throughout the observation period. For absent objects at any given time, fill the corresponding rows and columns with an arbitrary value (e.g., 0).

2. **Creating Inclusion Vectors**, indicating the presence (1) or absence (0) of each object for each time period. These vectors control which parts of the matrices to consider during mapping at any given time.

The `preprocessing` module's `expand_matrices()` function simplifies this process by automatically balancing the data and generating inclusion vectors.

Consider an extended TNIC dataset including Netflix, which only appears in the dataset after its IPO in 2002, resulting in unbalanced data. The following steps illustrate how to prepare and analyze this dataset:

1. Load and Balance the Data:

```
>>> from evomap.preprocessing import edgelist2matrices, expand_matrices
>>> S_t, labels = edgelist2matrices(
...     data_unbalanced,
...     score_var='score',
...     id_var_i='name1',
...     id_var_j='name2',
...     time_var='year')
>>> print(S_t[0].shape)   # Output: (9,9)
>>> S_t, inc_t, labels = expand_matrices(S_t, labels)
>>> print(S_t[0].shape)   # Output: (10,10)
>>> print(inc_t[0])       # Output: [1 1 1 1 1 1 1 1 1 0]
```

2. Transform Data and Fit EvoMap:

```
>>> D_t = [sim2diss(S, transformation='mirror') for S in S_t]
>>> init_t = [np.concatenate([cmds, np.array([[0,0]])], axis = 0) for cmds in cmds_t]
>>> X_t = EvoMDS(
...     alpha=.75,
...     p=2,
...     init=init_t,
...     mds_type='ordinal'
... ).fit_transform(D_t, inclusions=inc_t)
```

3. Visualize the Results:

```
>>> draw_map(X_t[0],
...     inclusions=inc_t[0],
...     label=labels,
...     show_axes=True)
>>> draw_map(X_t[-1],
...     inclusions=inc_t[-1],
...     label=labels,
...     show_axes=True)
```

Figure 10 shows the first and last snapshot of the generated sequence. As seen in the Figure, EvoMap has successfully mapped the unbalanced data. The inclusion vectors ensure that

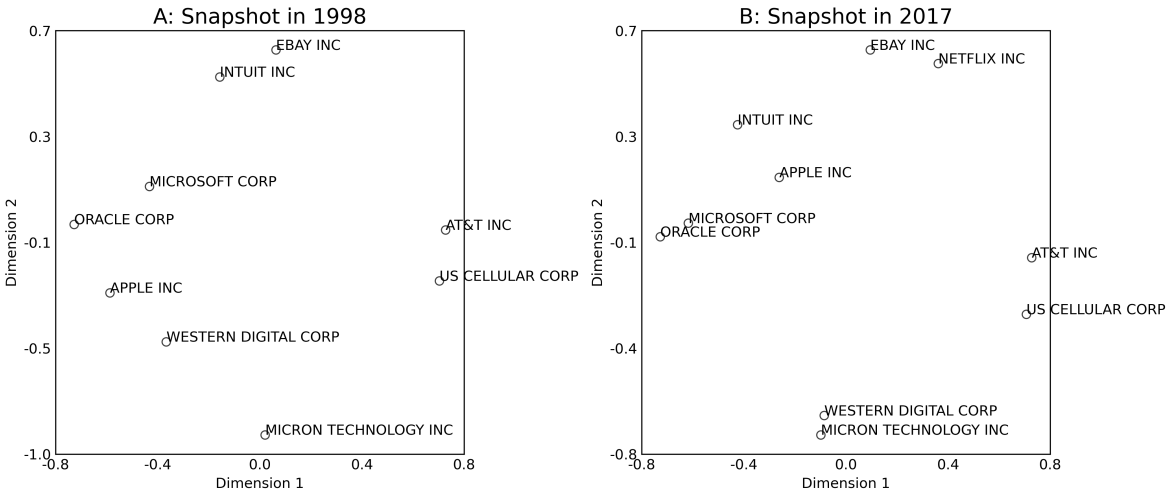

Figure 10: EvoMap Snapshots for 1998 and 2017, Using Unbalanced Data

EvoMap accurately reflects the entry and exit of firms: Netflix is not present on the left map corresponding to 1998, while it appears close to software-focused firms (eBay) on the right map corresponding to 2017.

6. Summary and Discussion

The **evomap** package, showcased in this paper, equips users with a comprehensive toolbox for dynamic mapping, enabling them to focus on their specific application rather than the method's implementation. The package's capabilities span data preparation, model fitting, visualization, and evaluation, supporting a wide range of potential applications.

While the package offers a broad range of functionalities, it also presents opportunities for future extensions. Currently, **evomap** integrates three core mapping methods (MDS, Sammon Mapping, and t-SNE). As outlined in section 3, however, its modular structure makes it easy to expand this set of methods. While the currently implemented methods are versatile, they come with specific requirements for the input data. For instance, all methods require complete observations for each time point. This means that while EvoMap can accomodate unbalanced data over time, it cannot accommodate incomplete data (e.g., due to missing relationships). At the same time, the implemented methods do not allow for asymmetric relationships, nor do they generate output other than Cartesian planes. As static mapping methods exist that can accommodate these requirements, like asymmetric or spherical MDS, adding them to the evomap package could further extend EvoMap's possible areas of application. Further, combining EvoMap with external preference analysis or property fitting could extend the resultant maps' interpretability.

To conlcude, we hope that the **evomap** package can aid users in applying the EvoMap framework, and that it can serve as a foundation for manifold applications of dynamic mapping in different domains.

Acknowledgments

This research was supported by the German Research Foundation, Deutsche Forschungsgemeinschaft (grant number SK 66/9-1).

The authors would like to thank Alexsey Pechnikov for invaluable support in optimizing the code for this project.

References

Borg I, Groenen PJ (2005). *Modern multidimensional scaling: Theory and applications*. 2. edition. Springer Science & Business Media, New York, NY. doi:https://doi.org/10.1007/0-387-28981-X.

Carroll JD, Arabie P (1998). "Multidimensional Scaling." In MH Birnbaum (ed.), *Measurement, Judgment and Decision Making*, Handbook of Perception and Cognition (Second Edition), pp. 179–250. Academic Press, San Diego, CA. doi:https://doi.org/10.1016/B978-012099975-0.50005-1.

Carroll JD, Chang JJ (1970). "Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition." *Psychometrika*, **35**(3), 283–319. doi:https://doi.org/10.1007/BF02310791.

Chen L, Buja A (2009). "Local multidimensional scaling for nonlinear dimension reduction, graph drawing, and proximity analysis." *Journal of the American Statistical Association*, **104**(485), 209–219. doi:https://doi.org/10.1198/jasa.2009.0111.

De Leeuw J, Mair P (2009). "Multidimensional scaling using majorization: SMACOF in R." *Journal of Statistical Software*, **31**(3), 1–30. doi:https://doi.org/10.18637/jss.v031.i03.

DeSarbo WS, Hoffman DL (1987). "Constructing MDS joint spaces from binary choice data: A multidimensional unfolding threshold model for marketing research." *Journal of Marketing Research*, **24**(1), 40–54. doi:https://doi.org/10.1177/002224378702400104.

DeSarbo WS, Manrai AK, Manrai LA (1994). "Latent class multidimensional scaling. A review of recent developments in the marketing and psychometric literature." In R Bagozzi (ed.), *Advanced Methods of Marketing Research*, pp. 190–222. Blackwell, Oxford.

Goodwill A, Meloy JR (2019). "Visualizing the relationship among indicators for lone actor terrorist attacks: Multidimensional scaling and the TRAP-18." *Behavioral Sciences & the Law*, **37**(5), 522–539. doi:https://doi.org/10.1002/bsl.2434.

Goslee SC, Urban DL (2007). "The ecodist package for dissimilarity-based analysis of ecological data." *Journal of Statistical Software*, **22**(7), 1–19. doi:https://doi.org/10.18637/jss.v022.i07.

Hebart MN, Zheng CY, Pereira F, Baker CI (2020). "Revealing the multidimensional mental representations of natural objects underlying human similarity judgements." *Nature Human Behaviour*, **4**(11), 1173–1185. doi:https://doi.org/10.1038/s41562-020-00951-3.

Hoberg G, Phillips G (2016). "Text-based network industries and endogenous product differentiation." *Journal of Political Economy*, **124**(5), 1423–1465. doi:https://doi.org/10.1086/688176.

Jacoby WG, Armstrong DA (2014). "Bootstrap confidence regions for multidimensional scaling solutions." *American Journal of Political Science*, **58**(1), 264–278. doi:https://doi.org/10.1111/ajps.12056.

Kenkel NC, Orlóci L (1986). "Applying metric and nonmetric multidimensional scaling to ecological studies: Some new results." *Ecology*, **67**(4), 919–928. doi:https://doi.org/10.2307/1939814.

Krijthe JH (2015). *Rtsne: T-distributed stochastic neighbor embedding using barnes-hut implementation*. R package version 0.17, URL https://github.com/jkrijthe/Rtsne.

Kruskal JB (1964a). "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis." *Psychometrika*, **29**(1), 1–27. doi:https://doi.org/10.1007/BF02289565.

Kruskal JB (1964b). "Nonmetric multidimensional scaling: A numerical method." *Psychometrika*, **29**(2), 115–129. doi:https://doi.org/10.1007/BF02289694.

Le S, Husson F (2008). "SensoMineR: A package for sensory data analysis." *Journal of Sensory Studies*, **23**(1), 14–25. doi:https://doi.org/10.1111/j.1745-459X.2007.00137.x.

Mair P, Groenen PJ, de Leeuw J (2022). "More on multidimensional scaling and unfolding in R: Smacof version 2." *Journal of Statistical Software*, **102**(10), 1–47. doi:https://doi.org/10.18637/jss.v102.i10.

Mane KK, Börner K (2004). "Mapping topics and topic bursts in PNAS." *Proceedings of the National Academy of Sciences*, **101**, 5287–5290. doi:https://doi.org/10.1073/pnas.0307626100.

Matthe M, Ringel DM, Skiera B (2023). "Mapping market structure evolution." *Marketing Science*, **42**(3), 589–613. doi:https://doi.org/10.1287/mksc.2022.1385.

McInnes L, Healy J, Melville J (2018). "Umap: Uniform manifold approximation and projection for dimension reduction." *arXiv preprint arXiv:1802.03426*. URL https://arxiv.org/abs/1802.03426.

Oksanen J, Simpson GL, Blanchet FG, Kindt R, Legendre P, Minchin PR, O'Hara R, Solymos P, Henry M, Stevens H, Szoecs E, Wagner H, Barbour M, Bedward M, Bolker B, Borcard D, Carvalho G, Chirico M, Caceres MD, Durand S, Beatriz H, Evangelista A, FitzJohn R, Friendly M, Furneaux B, Hannigan G, Hill MO, Lahti L, McGlinn D, Ouellette MH, Cunha ER, Smith T, Stier A, Braak CJT, Weedon J (2022). *Vegan: Community ecology package*. R package version 2.6.4, URL https://cran.r-project.org/web/packages/vegan/index.html.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V (2011). "Scikit-learn: Machine learning in Python." *Journal of Machine Learning Research*, **12**, 2825–2830.

Perera D (2023). *Sammon Mapping*. Python package version 0.0.2, URL https://pypi.org/project/sammon-mapping/.

Ringel DM, Skiera B (2016). "Visualizing asymmetric competition among more than 1,000 products using big search data." *Marketing Science*, **35**(3), 511–534. doi:https://doi.org/10.1287/mksc.2015.0950.

Saeed N, Nam H, Al-Naffouri TY, Alouini MS (2019). "A state-of-the-art survey on multidimensional scaling-based localization techniques." *IEEE Communications Surveys & Tutorials*, **21**(4), 3565–3583. doi:https://doi.org/10.1109/COMST.2019.2921972.

Sammon JW (1969). "A nonlinear mapping for data structure analysis." *IEEE Transactions on Computers*, **100**(5), 401–409. doi:https://doi.org/10.1109/T-C.1969.222678.

Shepard RN (1962a). "The analysis of proximities: Multidimensional scaling with an unknown distance function. I." *Psychometrika*, **27**(2), 125–140. doi:https://doi.org/10.1007/BF02289630.

Shepard RN (1962b). "The analysis of proximities: Multidimensional scaling with an unknown distance function. II." *Psychometrika*, **27**(3), 219–246. doi:https://doi.org/10.1007/BF02289621.

Torgerson WS (1958). *Theory and methods of scaling*. John Wiley, New York, NY.

van der Maaten L, Hinton G (2008). "Visualizing data using t-SNE." *Journal of Machine Learning Research*, **9**(11).

Venables WN, Ripley BD (2002). *Modern applied statistics with S*. Springer, New York, NY. doi:https://doi.org/10.1007/978-0-387-21706-2.

Affiliation:

Maximilian Matthe
Kelley School of Business, Indiana University
Department of Marketing
1309 E 10th St
47405 Bloomington, IN, USA
E-mail: mpmatthe@iu.edu
URL: https://mpmatthe.github.io

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd