The Journal of Open Source Software

PyOTC: A Python Package for Optimal Transition Coupling

Bongsoo Yi [1], **Yuning Pan** [2], **and Jay Hineman** [3]

1 Department of Statistics and Operations Research, University of North Carolina at Chapel Hill, Chapel Hill, NC, United States of America **2** Department of Mathematics and Statistics, Boston University, Boston, MA, United States of America **3** Applied Research Associates, Raleigh, NC, United States of America

DOI: 10.21105/joss.10576

Software

- Review
- Repository
- Archive

Editor: Wentao Ye

Reviewers:

- @HaoyuanHe0606
- @pebabion
- @r0hin

Submitted: 14 February 2026
Published: 27 June 2026

Summary

Recent scholarly work (O'Connor et al., 2022) introduced an extension of optimal transport that applies directly to stationary Markov processes. This extension enables the computation of meaningful distances between such processes, facilitating comparisons of networks and graphs in fields such as chemistry, biology, and social science. We provide a performant Python implementation of this method (O'Connor et al., 2022), along with interfaces for related network and graph problems (Yi et al., 2024). Our implementation is open source, tested, and integrates with the Python data science ecosystem.

State of the Field

Optimal transport (OT) has become a widely used computational framework across machine learning, statistics, and network science. A mature ecosystem of OT software already exists, most notably `POT` (Python Optimal Transport) (Flamary et al., 2021), which provides efficient solvers for classical optimal transport problems and their entropic approximations. However, optimal transport between *Markov processes*—specifically optimal transition coupling (OTC) introduced by O'Connor et al. (2022)—requires additional structure beyond standard OT formulations.

Two implementations of OTC currently exist in MATLAB (O'Connor, 2022; Yi, 2023). While these implementations supported the original research developments, they share limitations typical of the MATLAB ecosystem: they are not always freely accessible and lack integration with the broader Python-based scientific computing ecosystem. This makes it difficult to incorporate OTC methods into modern data science workflows.

`pyotc` addresses this gap by providing the first open-source Python implementation of the exact OTC formulation. The package builds on existing optimal transport infrastructure such as `POT` (Flamary et al., 2021), while implementing the additional algorithmic structure required for OTC. This allows researchers to apply OTC methods within the broader Python scientific computing ecosystem.

Software Design

The design of `pyotc` prioritizes modularity, interoperability with existing optimal transport infrastructure, and scalability to larger graph-based problems. A key design decision was to build on top of the `POT` library (Flamary et al., 2021) rather than implementing low-level optimal transport solvers from scratch. Leveraging `POT` allows `pyotc` to reuse highly optimized

algorithms such as network simplex and Sinkhorn iterations while focusing development effort on the algorithmic structure specific to optimal transition coupling.

The architecture closely mirrors the theoretical algorithm introduced in O'Connor et al. (2022). The OTC problem is solved using a policy iteration procedure consisting of alternating evaluation and improvement steps. This separation makes the implementation easy to understand and extend while allowing the improvement stage to reuse existing optimal transport solvers.

`pyotc` supports both exact and entropically regularized solution methods. The exact approach is useful for validating implementations and enabling further algorithmic development, while entropic regularization provides a scalable approximation that can handle larger problems more efficiently. The choice of the entropic regularization parameter affects convergence behavior and remains an active research topic in approximate optimal transport and related formulations such as the Schrödinger Bridge problem (Nutz, 2021; Peyré & Cuturi, 2020).

To support practical applications in network analysis, the implementation also provides optional sparse matrix representations that reduce memory usage and allow larger transition graphs to be handled efficiently. Integration with Python libraries such as `NetworkX` (Hagberg et al., 2008) enables workflows involving graph construction, analysis, and comparison. Future applications may also involve integration with machine learning libraries such as `scikit-learn` (Pedregosa et al., 2011).

Performance Comparison

To demonstrate the performance improvements of `pyotc` over existing MATLAB implementations, we benchmark our Python implementation against the original MATLAB code from O'Connor (2022). The benchmark evaluates the execution time on two randomly generated stochastic block models (SBMs), each consisting of 4 blocks with an intra-block connection probability of 0.9 and an inter-block connection probability of 0.1.

The empirical results, summarized in Table 1, show that `pyotc` achieves substantial speedups while yielding identical results to the MATLAB implementation. All benchmarks were executed on a machine equipped with a 2.3 GHz quad-core Intel Core i7 processor and 16 GB of RAM. The MATLAB benchmarks were executed using MATLAB R2025b, with the JIT compiler pre-warmed prior to measurement to ensure accurate timing results.

Table 1: Execution time comparison (in seconds) between the MATLAB implementation and `pyotc` across different network sizes.

Number of Nodes	20	32	40	60	80	100
MATLAB (s)	5.51	26.88	61.27	308.20	1246.13	3890.41
PyOTC (s)	0.60	3.21	9.60	54.73	322.44	1340.12

Statement of Need

Optimal transport has proven to be a valuable tool in data science and machine learning. One natural extension beyond probability distributions is to stochastic processes, particularly stationary finite-state Markov chains.

Recent work (O'Connor et al., 2022) developed theory and algorithms for computing optimal transition couplings between such processes. Transition couplings provide a constrained family of transport plans that match marginal distributions while preserving transition dynamics.

These methods have demonstrated practical value in applications such as network alignment and graph comparison (Hoang et al., 2025; Yi et al., 2024). However, practical access to

OTC algorithms remains limited. Our goal is to provide a practical, open-source Python implementation that allows researchers to apply these methods in real-world data science workflows. The package is designed to integrate naturally with Python-based scientific computing tools and to support experimentation with OTC algorithms in a wide range of network analysis problems.

Comparison with other available codes and lineage

The original MATLAB code for OTC was written by Kevin O'Connor and used for the work in (O'Connor et al., 2022); the code is available on GitHub (O'Connor, 2022). This MATLAB implementation was later extended and applied to network alignment in (Yi et al., 2024). The development of `pyotc` was initiated from these predecessors.

Additional related codes have been developed, primarily focusing on the entropic case. Notably, Calo et al. (2024) provide a variation of the Sinkhorn iteration described in (O'Connor et al., 2022), with an accompanying implementation available on GitHub (Calo, 2024). Another related effort is the differentiable extension of entropic OTC described in (Brugère et al., 2024) and implemented in (Brugère, 2024). In contrast to these entropic variants, `pyotc` addresses the exact case with a `POT`-powered implementation. Relying on this exact formulation, rather than entropic approximations, is essential for maintaining strict numerical precision in downstream tasks.

Research Impact Statement

The `pyotc` package serves a broad audience of researchers working with network-structured data across various domains—including computational chemistry, structural biology, neuroimaging, and social sciences-where comparing, aligning, and measuring distances between graphs or Markov chains is a fundamental task. While the core optimal transition coupling (OTC) methodology was initially introduced in (O'Connor et al., 2022) and applied to network alignment (Yi et al., 2024), `pyotc` lowers the barrier to entry by providing a robust, Python-native framework that integrates seamlessly with modern scientific pipelines.

The package has already demonstrated direct research impact, bridging the gap between theoretical frameworks and empirical applications. The structural properties and algorithms provided in `pyotc` have been directly utilized, extended, or cited by several recent studies in graph matching, Markov chain analysis, and network alignment, e.g., Hoang et al. (2025), Xiang et al. (2026), Calo et al. (2024), Brugère et al. (2024), Zeng et al. (2023).

Beyond literature-based impact, `pyotc` has garnered significant interest from the broader scientific community. The repository has received explicit inquiries and requests for code sharing from external research groups, particularly those specializing in network science and neuroimaging analysis, who seek to apply OTC to empirical brain connectivity data. By transitioning the ecosystem from legacy, closed-loop MATLAB implementations to an open-source, `POT`-backed Python library, `pyotc` enables domain scientists to easily plug network datasets into modern workflows (such as `NetworkX` or `scikit-learn`), facilitating interdisciplinary applications and accelerating the adoption of optimal transport methods in practical network analysis.

Features

Our implementation includes the tools needed to reproduce the examples from (O'Connor et al., 2022) and (Yi et al., 2024) in Python. It achieves strong performance by leveraging optimized optimal transport backends such as the network simplex implementation in `POT` (Flamary et al., 2021).

The pyotc code provides two approaches to solving the OTC problem. The *exact* solution procedure solves the underlying optimal transport problems via linear programming. The specialized *network simplex* algorithm from POT is used in the improvement step of policy iteration. In the evaluation step, a block linear system involving the transition matrix R and the cost vector c is solved. Alternatively, pyotc also provides an iterative method based on entropic regularization. This approach allows the method to scale to larger problems while leveraging the extensive optimal transport functionality available in POT.

Algorithm 1 summarizes the exact OTC solution procedure introduced by O'Connor et al. (2022).

Algorithm 1: Exact OTC

1. Initialize $R_0 = P \otimes Q$, Convergence tolerance τ
2. Set `converged = False`, $R = R_0$
3. **While** not `converged`:
    1. **Evaluate transition coupling**: $g, h = \text{evaluate}(R)$
    2. **Improve transition coupling**: $R_0 = R$, $R = \text{improve}(g, h)$
    3. **Check convergence**: $d = \|R - R_0\|$, $\text{converged} = d < \tau$
4. **Output**: R (an optimal transition coupling)

Examples

We provide a basic hello world example here. Our implementation is well documented and simple, consisting essentially of Python functions, which makes it easy to modify.

```
from pyotc import exact_otc
import numpy as np

P = np.array([[.5, .5], [.5, .5]])
Q = np.array([[0, 1], [1, 0]])
c = np.array([[1, 0], [0, 1]])

exp_cost, R, stat_dist = exact_otc(P, Q, c)
print("\nExact OTC cost between P and Q:", exp_cost)
```

Conclusion

pyotc provides a performant Python implementation for computing optimal transition couplings for stationary Markov chains and their associated graph structures. Optimal transition coupling is a clear opportunity to bring a novel computational tool to a wider audience through open-source software. By moving to an open ecosystem such as Python, we have produced an OTC code that is faster and, arguably, more capable than existing implementations.

As OTC is an active research topic, we believe there are significant opportunities to extend the work here. In this direction, we hope that this code will facilitate further explorations in both novel algorithms and more general implementations. For example, one could explore variations on the policy improvement and policy evaluation algorithms in terms of the stationary distribution (essentially a resolvent calculation). Implementation-wise, there are significant opportunities to provide additional interfaces to the Python ecosystem, including cheminformatics and bioinformatics resources such as RDKit (Landrum et al., 2025). pyotc also enables additional benchmarking studies.

AI Usage Disclosure

No AI-generated code was used in the development of the software. GPT 5.2 was used to help refine and edit the manuscript.

References

Brugère, T. (2024). *YusuLab/ot_markov_distances*. https://github.com/YusuLab/ot_markov_distances

Brugère, T., Wan, Z., & Wang, Y. (2024). Distances for Markov chains, and their differentiation. *International Conference on Algorithmic Learning Theory*, 282–336.

Calo, S. (2024). *SergioCalo/SVI*. https://github.com/SergioCalo/SVI

Calo, S., Jonsson, A., Neu, G., Schwartz, L., & Segovia-Aguas, J. (2024). Bisimulation metrics are optimal transport distances, and can be computed efficiently. *Advances in Neural Information Processing Systems*, *37*, 134345–134386. https://doi.org/10.52202/079017-4269

Flamary, R., Courty, N., Gramfort, A., Alaya, M. Z., Boisbunon, A., Chambon, S., Chapel, L., Corenflos, A., Fatras, K., Fournier, N., Gautheron, L., Gayraud, N. T. H., Janati, H., Rakotomamonjy, A., Redko, I., Rolet, A., Schutz, A., Seguy, V., Sutherland, D. J., … Vayer, T. (2021). POT: Python optimal transport. *Journal of Machine Learning Research*, *22*(78), 1–8. http://jmlr.org/papers/v22/20-451.html

Hagberg, A., Swart, P. J., & Schult, D. A. (2008). *Exploring network structure, dynamics, and function using NetworkX* (LA-UR-08-05495; LA-UR-08-5495). Los Alamos National Laboratory (LANL), Los Alamos, NM (United States). https://doi.org/10.25080/tcwv9851

Hoang, P. N., McGoff, K., Nobel, A. B., Xiang, Y., & Yi, B. (2025). *Optimal graph joining with applications to isomorphism detection and identification*. https://doi.org/10.48550/arXiv.2511.14862

Landrum, G., Tosco, P., Kelley, B., Rodriguez, R., Cosgrove, D., Vianello, R., sriniker, Gedeck, P., Jones, G., Kawashima, E., NadineSchneider, Nealschneider, D., Dalke, A., Swain, M., Cole, B., tadhurst-cdd, Turk, S., Savelev, A., Vaucher, A., … godin, guillaume. (2025). *Rdkit/rdkit: 2025_03_4 (Q1 2025) release*. Zenodo. https://doi.org/10.5281/zenodo.15773589

Nutz, M. (2021). Introduction to entropic optimal transport. *Lecture Notes, Columbia University*, *306*(19), 307.

O'Connor, K. (2022). *Oconnor-kevin/OTC*. https://github.com/oconnor-kevin/OTC

O'Connor, K., McGoff, K., & Nobel, A. B. (2022). Optimal Transport for Stationary Markov Chains via Policy Iteration. *Journal of Machine Learning Research*, *23*(45), 1–52. http://jmlr.org/papers/v23/21-0519.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830. https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

Peyré, G., & Cuturi, M. (2020). *Computational optimal transport* (No. arXiv:1803.00567). arXiv. https://doi.org/10.48550/arXiv.1803.00567

Xiang, Y., McGoff, K., & Nobel, A. B. (2026). *Graph disjointness with applications to reversible Markov chains*. https://doi.org/10.48550/arXiv.2603.02563

Yi, B. (2023). *Austinyi/NetOTC*. https://github.com/austinyi/NetOTC

Yi, B., O'Connor, K., McGoff, K., & Nobel, A. B. (2024). Alignment and comparison of directed networks via transition couplings of random walks. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, qkae085. https://doi.org/10.1093/jrsssb/qkae085

Zeng, Z., Zhang, S., Xia, Y., & Tong, H. (2023). PARROT: Position-aware regularized optimal transport for network alignment. *Proceedings of the ACM Web Conference 2023*, 372–382. https://doi.org/10.1145/3543507.3583357