A Space Efficient Approximation of Optimal Transport via Hierarchically Separated Trees

Ian Wu[1, 3], Charles Zhang[2, 3], Alina Ene[3]

Del Norte High School, 16601 Nighthawk Lane, San Diego, CA, 92127[1]; Great Oak High School, 32555 Deer Hollow Way, Temecula, CA 92592[2]; Boston University, Boston, MA 02215[3]

Introduction

- **Optimal Transport** (OT) is a well-studied optimization problem with applications through machine learning (domain adaptation, resource allocation, etc.)
  - OT seeks to minimize the cost of transporting mass between two probability distributions. In other words, OT seeks to minimize $\min_{\mathbf{P} \in \mathbb{R}^{n \times n}} \langle \mathbf{C}, \mathbf{P} \rangle$ where $\mathbf{C} \in \mathbb{R}^{n \times n}$ and **P** is a matrix representing mass transfer [1]
  - **Sinkhorn**, the most widely used OT algorithm, utilizes entropic regularization to efficiently compute an approximate solution in ***O(n²)*** time [2]
    - Sinkhorn must store the entire transport matrix in memory, however, using ***O(n²)*** space
  - **Hierarchical Refinement** (HiRef) attempts to solve this by grouping points in bijective couplings, but relies on several restrictive assumptions, limiting its applicability despite using ***O(n)*** space [3]
    - Assumes same number of source and target points that all have equal weight, does not permit mass splitting

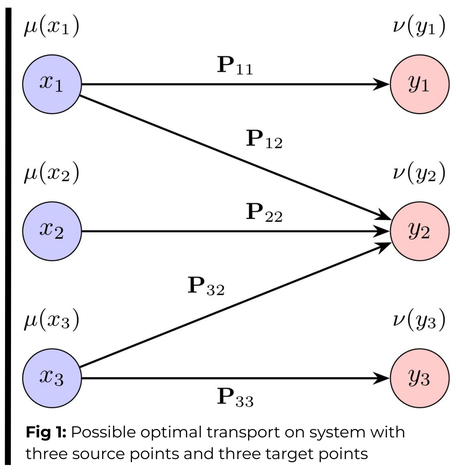


Fig 1: Possible optimal transport on system with three source points and three target points

- We introduce a new algorithm to approximate OT in ***O(nlogΔ)*** space, an improvement upon Sinkhorn, by grouping points using a Hierarchically Well-Separated Tree Embedding (HST), while accomodating aforementioned limitations.
  - We test the algorithm's efficiency and accuracy on several synthetic datasets, in addition to comparing it with Sinkhorn

Algorithm/Methods

Algorithm

Our algorithm begins by generating an **FRT Tree Embedding** [4] for the data points, with the following modifications:

- If a node is only associated with one point, we stop generating its children and assign a difference value representing its weight
- After generating a node's children, remove the set of nodes it represents from memory
- Iterate over only node-represented points when generating its children, instead of every point

Then, we calculate the cost of optimal transport by pushing mass towards the root node on the tree and maximizing lower-cost matchings farther from the root node.

Algorithm 2 Mass-Pushing Cost Calculation

procedure CALCULATE_COST(node, π)
  if node.branches $\neq \varnothing$ **then**
    sourceSet $\leftarrow \{ b \in \text{node.branches} \mid b.\text{difference} > 0 \}$
    targetSet $\leftarrow \{ b \in \text{node.branches} \mid b.\text{difference} < 0 \}$
    totalMass $\leftarrow \min(\sum s.\text{difference}, \sum(-t.\text{difference}))$
    deferred $\leftarrow [\,]$
    for all $p_1 \in$ sourceSet **do**
      for all $p_2 \in$ targetSet **do**
        $\text{mass}' \leftarrow \text{totalMass} \times \left(\frac{p_1.\text{difference}}{\sum \text{sourceSet}}\right) \times \left(\frac{-p_2.\text{difference}}{\sum \text{targetSet}}\right)$
        deferred.**append**(p_1, p_2, mass')
    for all $(p_1, p_2, m) \in$ deferred **do**
      if p_1.leaf $\wedge$ p_2.leaf **then**
        node.cost $\leftarrow$ node.cost + dist$\pi[p_1], \pi[p_2] \times m$
        $p_1.\text{difference} \leftarrow p_1.\text{difference} - m$
        $p_2.\text{difference} \leftarrow p_2.\text{difference} + m$
      else if p_1.branch $\wedge$ p_2.leaf **then**
        $p_2.\text{difference} \leftarrow p_2.\text{difference} + m$
        PUSH_MASS(p_1, m, **true**, p_2, π)
      else if p_2.branch $\wedge$ p_1.leaf **then**
        $p_1.\text{difference} \leftarrow p_1.\text{difference} - m$
        PUSH_MASS(p_2, m, **false**, p_1, π)
      else
        node.cost $\leftarrow$ node.cost + PUSH_MASS_BETWEEN(node, p_1, p_2, m, π)
    for all $b \in$ node.branches **do**
      CALCULATE_COST(b, π)

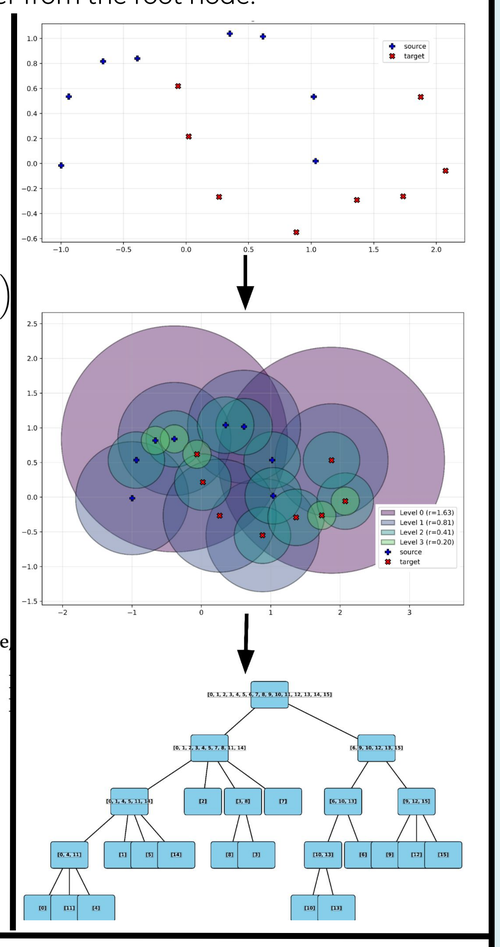


Fig. 2 (right): FRT Tree Embedding algorithm ran on a small two moons dataset. FRT greedily groups points using the ball function, as demonstrated by the middle image. The resulting tree (bottom image) groups nearby points into clusters with low inter-transport costs.

Comparison

- We compare our algorithm to Sinkhorn using the Python OT library with ε = 0.04 × mean(C)
- We test on a point cloud of n uniformly sampled source and target points with random Gaussian weights.
- We use the Earth Mover's Distance (EMD) as a baseline for approximation measurement

Results

Theoretical Results

- *Space:* Our algorithm avoids instantiating a cost matrix and instead utilize a tree, which contains every point in each of ***O(logΔ)*** layers, resulting in ***O(nlogΔ)*** space complexity
- *Time:* The time bottleneck of the algorithm is generating the FRT tree, which runs in ***O(n²)*** time.
- *Accuracy:* By using an FRT tree approximation, we incur a distortion of ***O(logn)*** such that

$$\mathbb{E}[c_T(u,v)] \leq O(\log n) c(u,v)$$

however, we demonstrate through tests that the average error within the cost approximation is significantly less than ***O(logn)***

Method	Time	Space	Loss	Caveats
EMD	$O(n^3 \log n)$	$O(n^2)$	≈ 0	Computationally expensive.
Sinkhorn	$O(n^2/\varepsilon^2)$	$O(n^2)$	$\leq +\varepsilon$	$\varepsilon \to 0$ incurs instability/slowdown.
HiRef	$O(n \log n)$	$O(n)$	≈ 0	Only solves for Monge maps.
HSTOT	$O(n^2)$	$O(n \log \Delta)$	$\leq \times O(\log n)$	Inaccurate for certain datasets.

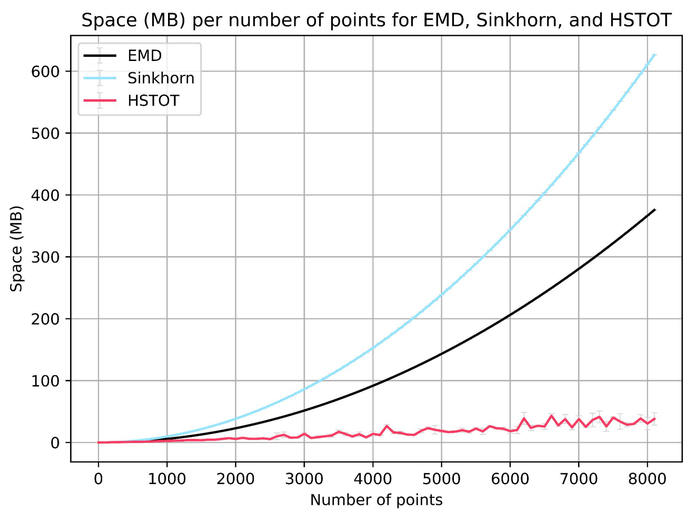
Table 1: Comparison of OT algorithms.

Empirical Results

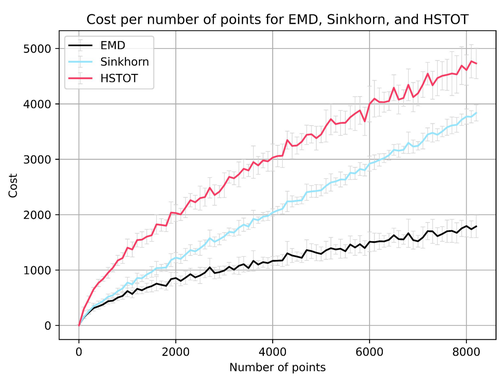
Fig. 3 (right): Space comparison between EMD, Sinkhorn, and our algorithm. EMD and Sinkhorn both use quadratic space, while our algorithm runs with significantly less speed.

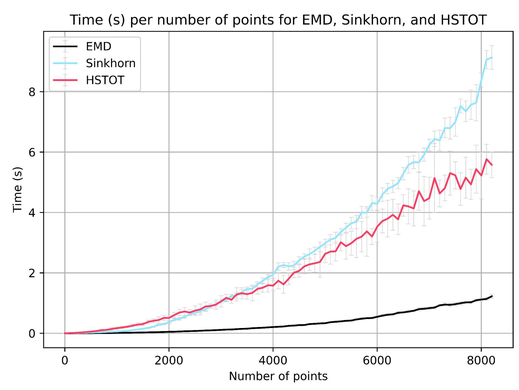
Fig. 4 (bottom-left): Accuracy comparison between EMD, Sinkhorn, and our algorithm. Our algorithm provides a less accurate approximate than Sinkhorn, but when compared to actual costs (EMD), both algorithms' errors increase at the same rate.

Fig. 5 (bottom-right): Time comparison between EMD, Sinkhorn, and our algorithm. Our algorithm and Sinkhorn have comparable runtime. EMD has a cubic time complexity and is typically signficantly slower in comparison, except in this case POT's EMD algorithm is implemented in C++ while Sinkhorn and ours are written in python, resulting in faster runtime.



Note: In Fig 3, 4, and 5, each algorithm was ran on 10 different randomly generated point clouds and weights, and the resulting measures were subsequently averaged to obtain means and standard deviations on the figures.





Discussion

Our algorithm attains a significantly lower space complexity but sacrifices accuracy when compared to Sinkhorn. Both algorithms have comparable speed.

Extensions

Unlike HiRef, our algorithm does not make restrictive assumptions about the data points and therefore can be applied to

- Partial Optimal Transport
  - If the source and target weights are not equivalent, leftover mass will be represented at the root node
- Non-Euclidean distance metrics
  - Does not rely on Euclidean distance, algorithm can be applied to other metrics as well

Future Work

- On some datasets (i.e. moon and blob), FRT may generate a tree with source and target points completely grouped together
  - Modifications could be made to FRT to avoid non-ideal groupings
- Testing on non-synthetic datasets

Acknowledgements

I would like to thank Dr. Alina Ene for her guidance and support throughout this project, who without this project would not have been possible. I would also like to thank the RISE program and BU for making this opportunity possible.

References

[1] L. Kantorovitch. On the translocation of masses. *Management Science*, 5(1):1–4, 1958.
[2] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transportation distances. 2013.
[3] Peter Halmos, Julian Gold, Xinhao Liu, and Benjamin Raphael. Hierarchical refinement: Optimal transport to infinity and beyond. *In Forty-second International Conference on Machine Learning*, 2025.
[4] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004. Special Issue on STOC 2003.