A. Complexity analysis

Sparse Sinkhorn. A common way of achieving a high p_1 and low p_2 in LSH is via the AND-OR construction. In this scheme we calculate $B \cdot r$ hash functions, divided into B sets (hash bands) of r hash functions each. A pair of points is considered as neighbors if any hash band matches completely. Calculating the hash buckets for all points with b hash buckets per function scales as $\mathcal{O}((n+m)dBbr)$ for the hash functions we consider. As expected, for the tasks and hash functions we investigated we obtain approximately m/b^r and n/b^r neighbors, with b^r hash buckets per band. Using this we can fix the number of neighbors to a small, constant β in expectation with $b^r = \min(n,m)/\beta$. We thus obtain a sparse cost matrix $\boldsymbol{C}^{\text{sp}}$ with $\mathcal{O}(\max(n,m)\beta)$ non-infinite values and can calculate $\boldsymbol{s}$ and $\boldsymbol{t}$ in linear time $\mathcal{O}(N_{\text{sink}} \max(n,m)\beta)$, where $N_{\text{sink}} \leq 2 + \frac{-4\ln(\min_{i,j}\{\tilde{\boldsymbol{K}}_{ij} | \tilde{\boldsymbol{K}}_{ij}>0\} \min_{i,j}\{\boldsymbol{p}_i, \boldsymbol{q}_j\})}{\varepsilon}$ (see Theorem 4) denotes the number of Sinkhorn iterations. Calculating the hash buckets with $r = \frac{\log \min(n,m) - \log \beta}{\log b}$ takes $\mathcal{O}((n+m)dBb(\log\min(n,m) - \log\beta)/\log b)$. Since B, b, and β are small, we obtain roughly log-linear scaling with the number of points overall, i.e. $\mathcal{O}(n \log n)$ for $n \approx m$.

LCN-Sinkhorn. Both choosing landmarks via k-means++ sampling and via k-means with a fixed number of iterations have the same runtime complexity of $\mathcal{O}((n+m)ld)$. Precomputing $\boldsymbol{W}$ can be done in time $\mathcal{O}(nl^2 + l^3)$. The low-rank part of updating the vectors $\boldsymbol{s}$ and $\boldsymbol{t}$ can be computed in $\mathcal{O}(nl + l^2 + lm)$, with l chosen constant, i.e. independently of n and m. Since sparse Sinkhorn with LSH has a log-linear runtime we again obtain log-linear overall runtime for LCN-Sinkhorn.

B. Limitations

Sparse Sinkhorn. Using a sparse approximation for $\boldsymbol{K}$ works well in the common case when the regularization parameter λ is low and the cost function varies enough between data pairs, such that the transport plan $\boldsymbol{P}$ resembles a sparse matrix. However, it can fail if the cost between pairs is very similar or the regularization is very high, if the dataset contains many hubs, i.e. points with a large number of neighbors, or if the distributions $\boldsymbol{p}$ or $\boldsymbol{q}$ are spread very unevenly. Furthermore, sparse Sinkhorn can be too unstable to train a model from scratch, since randomly initialized embeddings often have no close neighbors (see Sec. 8). Note also that LSH requires the cost function to be associated with a metric space, while regular Sinkhorn can be used with arbitrary costs.

Note that we are only interested in an approximate solution with finite error ε. We therefore do not need the kernel matrix to be fully indecomposable or have total support, which would be necessary and sufficient for a unique (up to a scalar factor) and exact solution, respectively (Sinkhorn & Knopp, 1967). However, the sparse approximation is not guaranteed to have support (Def. 1), which is necessary and sufficient for the Sinkhorn algorithm to converge. The approximated matrix is actually very likely not to have support if we use one LSH bucket per sample. This is due to the non-quadratic block structure resulting from every point only having non-zero entries for points in the other data set that fall in the same bucket. We can alleviate this problem by using unbalanced OT, as proposed in Sec. 6, or (empirically) the AND-OR construction. We can also simply choose to ignore this as long as we limit the maximum number of Sinkhorn iterations. On the 3D point cloud and random data experiments we indeed ignored this issue and actually observed good performance. Experiments with other LSH schemes and the AND-OR construction showed no performance improvement despite the associated cost matrices having support. Not having support therefore seems not to be an issue in practice, at least for the data we investigated.

LCN-Sinkhorn. The LCN approximation is guaranteed to have support due to the Nyström part. Other weak spots of sparse Sinkhorn, such as very similar cost between pairs, high regularization, or data containing many hubs, are also usually handled well by the Nyström part of LCN. Highly concentrated distributions $\boldsymbol{p}$ and $\boldsymbol{q}$ can still have adverse effects on LCN-Sinkhorn. We can compensate for these by sampling landmarks or neighbors proportional to each point's probability mass.

The Nyström part of LCN also has its limits, though. If the regularization parameter is low or the cost function varies greatly, we observed stability issues (over- and underflows) of the Nyström approximation because of the inverse $\boldsymbol{A}^{-1}$, which cannot be calculated in log-space. The Nyström approximation furthermore is not guaranteed to be non-negative, which can lead to catastrophic failures if the matrix product in Eq. (2) becomes negative. In these extreme cases we also observed catastrophic elimination with the correction $\boldsymbol{K}^{\text{sp}}_{\Delta}$. Since a low entropy regularization essentially means that optimal transport is very local, we recommend using sparse Sinkhorn in these scenarios. This again demonstrates the complementarity of the sparse approximation and Nyström: In cases where one fails we can often resort to the other.

C. Proof of Theorem 1

By linearity of expectation we obtain

$$\begin{aligned}\mathbb{E}[\boldsymbol{K}_{i,i_k} - \boldsymbol{K}_{\text{Nys},i,i_k}] &= \mathbb{E}[\boldsymbol{K}_{i,i_k}] - \mathbb{E}[\boldsymbol{K}_{\text{Nys},i,i_k}] \\ &= \mathbb{E}[e^{-\delta_k/\lambda}] - \mathbb{E}[\boldsymbol{K}_{\text{Nys},i,i_k}]\end{aligned} \tag{20}$$

with the distance to the kth-nearest neighbor δ_k. Note that without loss of generality we can assume unit manifold volume and obtain the integral resulting from the first expectation as (ignoring boundary effects that are exponentially small in n, see Percus & Martin (1998))

$$\mathbb{E}[e^{-\delta_k/\lambda}] \approx \frac{n!}{(n-k)!(k-1)!} \int_0^{\frac{((d/2)!)^{1/d}}{\sqrt{\pi}}} e^{-r/\lambda} V_d(r)^{k-1} (1-V_d(r))^{n-k} \frac{\partial V_d(r)}{\partial r} \,\mathrm{d}r, \tag{21}$$

with the volume of the d-ball

$$V_d(r) = \frac{\pi^{d/2} r^d}{(d/2)!}. \tag{22}$$

Since this integral does not have an analytical solution we can either calculate it numerically or lower bound it using Jensen's inequality (again ignoring exponentially small boundary effects)

$$\mathbb{E}[e^{-\delta_k/\lambda}] \geq e^{-\mathbb{E}[\delta_k]/\lambda} \approx \exp\left(-\frac{((d/2)!)^{1/d}}{\sqrt{\pi}\lambda} \frac{(k-1+1/d)!}{(k-1)!} \frac{n!}{(n+1/d)!} \right). \tag{23}$$

To upper bound the second expectation $\mathbb{E}[\boldsymbol{K}_{\text{Nys},i,i_k}]$ we now denote the distance between two points by $r_{ia} = \|\boldsymbol{x}_{\text{p}i} - \boldsymbol{x}_a\|_2$, the kernel by $k_{ia} = e^{-r_{ia}/\lambda}$ and the inter-landmark kernel matrix by $\boldsymbol{K}_\text{L}$. We first consider

$$\begin{aligned} p(\boldsymbol{x}_j \mid \boldsymbol{x}_j \text{ is } k\text{th-nearest neighbor}) &= \\ &= \int p(\delta_k = r_{ij} \mid \boldsymbol{x}_i, \boldsymbol{x}_j) p(\boldsymbol{x}_i) p(\boldsymbol{x}_j) \,\mathrm{d}\boldsymbol{x}_i \\ &= \int p(\delta_k = r_{ij} \mid r_{ij}) p(r_{ij} \mid \boldsymbol{x}_j) \,\mathrm{d}r_{ij}\, p(\boldsymbol{x}_j) \\ &= \int p(\delta_k = r_{ij} \mid r_{ij}) p(r_{ij}) \,\mathrm{d}r_{ij}\, p(\boldsymbol{x}_j) \\ &= \int p(\delta_k = r_{ij}) \,\mathrm{d}r_{ij}\, p(\boldsymbol{x}_j) \\ &= p(\boldsymbol{x}_j) = p(\boldsymbol{x}_i), \end{aligned} \tag{24}$$

where the third step is due to the uniform distribution. Since landmarks are more than $2R$ apart we can approximate

$$\boldsymbol{K}_\text{L}^{-1} = (I_l + \mathbf{1}_{l\times l}\mathcal{O}(e^{-2R/\lambda}))^{-1} = I_n - \mathbf{1}_{l\times l}\mathcal{O}(e^{-2R/\lambda}), \tag{25}$$

where $\mathbf{1}_{l\times l}$ denotes the constant 1 matrix, with the number of landmarks l. We can now use (1) the fact that landmarks are arranged apriori, (2) Hölder's inequality, (3) Eq. (24), and (4) Eq. (25) to obtain

$$\begin{aligned} \mathbb{E}[\boldsymbol{K}_{\text{Nys},i,i_k}] &= \mathbb{E}\left[\sum_{a=1}^{l} \sum_{b=1}^{l} k_{ia} (\boldsymbol{K}_\text{L}^{-1})_{ab} k_{i_k b} \right] \\ &\overset{(1)}{=} \sum_{a=1}^{l} \sum_{b=1}^{l} (\boldsymbol{K}_\text{L}^{-1})_{ab} \mathbb{E}[k_{ia} k_{i_k b}] \\ &\overset{(2)}{\leq} \sum_{a=1}^{l} \sum_{b=1}^{l} (\boldsymbol{K}_\text{L}^{-1})_{ab} \mathbb{E}[k_{ia}^2]^{1/2} \mathbb{E}[k_{i_k b}^2]^{1/2} \\ &\overset{(3)}{=} \sum_{a=1}^{l} \sum_{b=1}^{l} (\boldsymbol{K}_\text{L}^{-1})_{ab} \mathbb{E}[k_{ia}^2]^{1/2} \mathbb{E}[k_{ib}^2]^{1/2} \\ &\overset{(4)}{=} \sum_{a=1}^{l} \mathbb{E}[k_{ia}^2] - \mathcal{O}(e^{-2R/\lambda}). \end{aligned} \tag{26}$$

Since landmarks are more than $2R$ apart we have $V_\mathcal{M} \geq l V_d(R)$, where $V_\mathcal{M}$ denotes the volume of the manifold. Assuming Euclideanness in $V_d(R)$ we can thus use the fact that data points are uniformly distributed to obtain

$$\begin{aligned} \mathbb{E}[k_{ia}^2] &= \mathbb{E}[e^{-2r_{ia}/\lambda}] \\ &= \frac{1}{V_\mathcal{M}} \int e^{-2r/\lambda} \frac{\partial V_d(r)}{\partial r} \,\mathrm{d}r \\ &\leq \frac{1}{l V_d(R)} \int e^{-2r/\lambda} \frac{\partial V_d(r)}{\partial r} \,\mathrm{d}r \\ &= \frac{1}{l V_d(R)} \int_0^R e^{-2r/\lambda} \frac{\partial V_d(r)}{\partial r} \,\mathrm{d}r + \mathcal{O}(e^{-2R/\lambda}) \\ &= \frac{d}{l R^d} \int_0^R e^{-2r/\lambda} r^{d-1} \,\mathrm{d}r + \mathcal{O}(e^{-2R/\lambda}) \\ &= \frac{d\,(\Gamma(d) - \Gamma(d, 2R/\lambda))}{l (2R/\lambda)^d} + \mathcal{O}(e^{-2R/\lambda}) \end{aligned} \tag{27}$$

and finally

$$\begin{aligned} \mathbb{E}[\boldsymbol{K}_{\text{Nys},i,i_k}] &\leq \sum_{a=1}^{l} \mathbb{E}[k_{ia}^2] - \mathcal{O}(e^{-2R/\lambda}) \\ &\leq \frac{d\,(\Gamma(d) - \Gamma(d, 2R/\lambda))}{(2R/\lambda)^d} + \mathcal{O}(e^{-2R/\lambda}). \end{aligned} \tag{28}$$

□

D. Proof of Theorem 2

We first prove two lemmas that will be useful later on.

Lemma A. *Let $\tilde{\boldsymbol{K}}$ be the Nyström approximation of the similarity matrix $\boldsymbol{K}_{ij} = e^{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2/\lambda}$, with all Nyström landmarks being at least D apart and data samples being* *no more than r away from its closest landmark. Then*

$$\tilde{\boldsymbol{K}}_{ij} = \tilde{\boldsymbol{K}}_{ij}^{\text{2L}} + \mathcal{O}(e^{-2\max(D-r, D/2)/\lambda}), \tag{29}$$

where $\tilde{\boldsymbol{K}}^{\text{2L}}$ denotes the Nyström approximation using only the two landmarks closest to the points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

Proof. We denote the landmarks closest to the two points i and j with the indices a and b, or jointly with $\mathbb{A}$, and all other landmarks with $\mathbb{C}$. We furthermore denote the kernel between the point i and the point a as $k_{ia} = e^{-\|\boldsymbol{x}_a - \boldsymbol{x}_j\|_2/\lambda}$ and the vector of kernels between a set of points $\mathbb{A}$ and a point i as $\boldsymbol{k}_{\mathbb{A}i}$.

We can split up $\boldsymbol{A}^{-1}$ used in the Nyström approximation

$$\tilde{\boldsymbol{K}} = \boldsymbol{U}\boldsymbol{A}^{-1}\boldsymbol{V}, \tag{30}$$

where $\boldsymbol{A}_{cd} = k_{cd}$, $\boldsymbol{U}_{ic} = k_{ic}$, and $\boldsymbol{V}_{dj} = k_{dj}$, into relevant blocks via

$$\begin{aligned} \boldsymbol{A}^{-1} &= \begin{pmatrix} \boldsymbol{A}_{\text{2L}} & \boldsymbol{B} \\ \boldsymbol{B}^T & \boldsymbol{A}_{\text{other}} \end{pmatrix}^{-1} \\ &= \begin{pmatrix} \boldsymbol{A}_{\text{2L}}^{-1} + \boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B}(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1}\boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1} & -\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B}(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1} \\ -(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1}\boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1} & (\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1} \end{pmatrix}, \end{aligned} \tag{31}$$

where $\boldsymbol{A}/\boldsymbol{A}_{\text{2L}} = \boldsymbol{A}_{\text{other}} - \boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B}$ denotes the Schur complement. We can thus write the entries of the Nyström approximation as

$$\begin{aligned} \tilde{\boldsymbol{K}}_{ij} &= \boldsymbol{k}_{\mathbb{A}i}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{k}_{\mathbb{A}j} \\ &\quad + \boldsymbol{k}_{\mathbb{A}i}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B}(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1}\boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{k}_{\mathbb{A}j} \\ &\quad - \boldsymbol{k}_{\mathbb{A}i}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B}(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1}\boldsymbol{k}_{\mathbb{C}j} \\ &\quad - \boldsymbol{k}_{\mathbb{C}i}^T(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1}\boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{k}_{\mathbb{A}j} \\ &\quad + \boldsymbol{k}_{\mathbb{C}i}^T(\boldsymbol{A}/\boldsymbol{A}_{\text{2L}})^{-1}\boldsymbol{k}_{\mathbb{C}j} \\ &= \tilde{\boldsymbol{K}}_{ij}^{\text{2L}} + (\boldsymbol{k}_{\mathbb{C}i}^T - \boldsymbol{k}_{\mathbb{A}i}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B}) \\ &\quad (\boldsymbol{A}_{\text{other}} - \boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B})^{-1} \\ &\quad (\boldsymbol{k}_{\mathbb{C}j} - \boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{k}_{\mathbb{A}j}). \end{aligned} \tag{32}$$

Interestingly, the difference to $\tilde{\boldsymbol{K}}_{ij}^{\text{2L}}$ is again a Nyström approximation where each factor is the difference between the correct kernel (e.g. $\boldsymbol{k}_{\mathbb{C}j}$) and the previous Nyström approximation of this kernel (e.g. $\boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{k}_{\mathbb{A}j}$).

We next bound the inverse, starting with

$$\begin{aligned} \boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B} &= \begin{pmatrix} \boldsymbol{k}_{\mathbb{C}a} & \boldsymbol{k}_{\mathbb{C}b} \end{pmatrix} \frac{1}{1-k_{ab}^2} \begin{pmatrix} 1 & -k_{ab} \\ -k_{ab} & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{k}_{\mathbb{C}a}^T \\ \boldsymbol{k}_{\mathbb{C}b}^T \end{pmatrix} \\ &= \frac{1}{1-k_{ab}^2}\left(\boldsymbol{k}_{\mathbb{C}a}\boldsymbol{k}_{\mathbb{C}a}^T - k_{ab}\boldsymbol{k}_{\mathbb{C}a}\boldsymbol{k}_{\mathbb{C}b}^T - k_{ab}\boldsymbol{k}_{\mathbb{C}b}\boldsymbol{k}_{\mathbb{C}a}^T + \boldsymbol{k}_{\mathbb{C}b}\boldsymbol{k}_{\mathbb{C}b}^T\right) \\ &= \mathbf{1}_{l-2\times l-2}(1 + \mathcal{O}(e^{-2D/\lambda})) \cdot 4\mathcal{O}(e^{-2D/\lambda}) \\ &= \mathbf{1}_{l-2\times l-2}\mathcal{O}(e^{-2D/\lambda}), \end{aligned} \tag{33}$$

where $\mathbf{1}_{l-2\times l-2}$ denotes the constant 1 matrix, with the number of landmarks l. The last steps use the fact that landmarks are more than D apart and $0 \le k \le 1$ for all k. For this reason we also have $\boldsymbol{A}_{\text{other}} = I_{l-2} + \mathbf{1}_{l-2\times l-2}\mathcal{O}(e^{-D/\lambda})$ and can thus use the Neumann series to obtain

$$\begin{aligned} (\boldsymbol{A}_{\text{other}} - \boldsymbol{B}^T\boldsymbol{A}_{\text{2L}}^{-1}\boldsymbol{B})^{-1} &= (I_{l-2} + \mathbf{1}_{l-2\times l-2}\mathcal{O}(e^{-D/\lambda}))^{-1} \\ &= I_{l-2} - \mathbf{1}_{l-2\times l-2}\mathcal{O}(e^{-D/\lambda}). \end{aligned} \tag{34}$$

We can analogously bound the other terms in Eq. (32) to obtain

$$\begin{aligned} \tilde{\boldsymbol{K}}_{ij} &= \tilde{\boldsymbol{K}}_{ij}^{\text{2L}} + (\boldsymbol{k}_{\mathbb{C}i}^T - \mathbf{1}_{1\times l-2}\mathcal{O}(e^{-D/\lambda})) \\ &\quad (I_{l-2} - \mathbf{1}_{l-2\times l-2}\mathcal{O}(e^{-D/\lambda})) \\ &\quad (\boldsymbol{k}_{\mathbb{C}j} - \mathbf{1}_{l-2\times 1}\mathcal{O}(e^{-D/\lambda})) \\ &\overset{(1)}{=} \tilde{\boldsymbol{K}}_{ij}^{\text{2L}} + \boldsymbol{k}_{\mathbb{C}i}^T\boldsymbol{k}_{\mathbb{C}j} + \mathcal{O}(e^{-(D+\max(D-r,D/2))/\lambda}) \\ &= \tilde{\boldsymbol{K}}_{ij}^{\text{2L}} + \sum_{\substack{1\le k\le l \\ k\ne a,b}} e^{-(\|\boldsymbol{x}_i - \boldsymbol{x}_k\|_2 + \|\boldsymbol{x}_k - \boldsymbol{x}_j\|_2)/\lambda} \\ &\quad + \mathcal{O}(e^{-(D+\max(D-r,D/2))/\lambda}) \\ &\overset{(2)}{\le} \tilde{\boldsymbol{K}}_{ij}^{\text{2L}} + de^{-2\max(D-r,D/2)/\lambda} \\ &\quad + \mathcal{O}(e^{-\max(2(D-r),(1+\sqrt{3})D/2)/\lambda}) \\ &= \tilde{\boldsymbol{K}}_{ij}^{\text{2L}} + \mathcal{O}(e^{-2\max(D-r,D/2)/\lambda}), \end{aligned} \tag{35}$$

where d denotes the dimension of $\boldsymbol{x}$. Step (1) follows from the fact that any points' second closest landmarks must be at least $\max(D-r, D/2)$ away (since landmarks are at least D apart). This furthermore means that any point can have at most d second closest landmarks at this distance, which we used in step (2). □

Lemma B. *Let $\tilde{\boldsymbol{K}}$ be the Nyström approximation of the similarity matrix $\boldsymbol{K}_{ij} = e^{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2/\lambda}$. Let $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ be data points with equal L_2 distance r_i and r_j to all l landmarks, which have the same distance $\Delta > 0$ to each other. Then*

$$\tilde{\boldsymbol{K}}_{ij} = \frac{le^{-(r_i+r_j)/\lambda}}{1 + (l-1)e^{-\Delta/\lambda}} \tag{36}$$

Proof. The inter-landmark distance matrix is

$$\boldsymbol{A} = e^{-\Delta/\lambda}\mathbf{1}_{l\times l} + (1 - e^{-\Delta/\lambda})\boldsymbol{I}_l, \tag{37}$$

where $\mathbf{1}_{l\times l}$ denotes the constant 1 matrix. Using the identity

$$(b\mathbf{1}_{n\times n} + (a-b)\boldsymbol{I}_n)^{-1} = \frac{-b}{(a-b)(a+(n-1)b)}\mathbf{1}_{n\times n} + \frac{1}{a-b}\boldsymbol{I}_n \tag{38}$$

we can compute

$$\begin{aligned}\tilde{\boldsymbol{K}}_{ij} &= \boldsymbol{U}_{i,:}\boldsymbol{A}^{-1}\boldsymbol{V}_{:,j}\\ &= \begin{pmatrix} e^{-r_i/\lambda} & e^{-r_i/\lambda} & \cdots \end{pmatrix} \left(\frac{-e^{-\Delta/\lambda}}{(1-e^{-\Delta/\lambda})(1+(l-1)e^{-\Delta/\lambda})} \mathbf{1}_{l\times l} + \frac{1}{1-e^{-\Delta/\lambda}} \boldsymbol{I}_l \right) \begin{pmatrix} e^{-r_j/\lambda} \\ e^{-r_j/\lambda} \\ \vdots \end{pmatrix}\\ &= \frac{e^{-(r_i+r_j)/\lambda}}{1-e^{-\Delta/\lambda}} \left(\frac{-l^2 e^{-\Delta/\lambda}}{1+(l-1)e^{-\Delta/\lambda}} + l \right) = \frac{e^{-(r_i+r_j)/\lambda}}{1-e^{-\Delta/\lambda}} \frac{l - l e^{-\Delta/\lambda}}{1+(l-1)e^{-\Delta/\lambda}}\\ &= \frac{l e^{-(r_i+r_j)/\lambda}}{1+(l-1)e^{-\Delta/\lambda}}.\end{aligned} \tag{39}$$

□

Moving on to the theorem, first note that it analyzes the maximum error realizable under the given constraints, not an expected error. $\boldsymbol{K}^{\text{sp}}$ is correct for all pairs inside a cluster and 0 otherwise. We therefore obtain the maximum error by considering the closest possible pair between clusters. By definition, this pair has distance $D - 2r$ and thus

$$\max_{\boldsymbol{x}_{\text{p}i}, \boldsymbol{x}_{\text{q}j}} \boldsymbol{K} - \boldsymbol{K}^{\text{sp}} = e^{-(D-2r)/\lambda} \tag{40}$$

LCN is also correct for all pairs inside a cluster, so we again consider the closest possible pair $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ between clusters. We furthermore use Lemma A to only consider the landmarks of the two concerned clusters, adding an error of $\mathcal{O}(e^{-2(D-r)/\lambda})$, since $r \ll D$. Hence,

$$\begin{aligned}\boldsymbol{K}^{\text{2L}}_{\text{LCN},ij} &= \begin{pmatrix} e^{-r/\lambda} & e^{-(D-r)/\lambda} \end{pmatrix} \begin{pmatrix} 1 & e^{-D/\lambda} \\ e^{-D/\lambda} & 1 \end{pmatrix}^{-1} \begin{pmatrix} e^{-(D-r)/\lambda} \\ e^{-r/\lambda} \end{pmatrix}\\ &= \frac{1}{1-e^{-2D/\lambda}} \begin{pmatrix} e^{-r/\lambda} & e^{-(D-r)/\lambda} \end{pmatrix} \begin{pmatrix} 1 & -e^{-D/\lambda} \\ -e^{-D/\lambda} & 1 \end{pmatrix} \begin{pmatrix} e^{-(D-r)/\lambda} \\ e^{-r/\lambda} \end{pmatrix}\\ &= \frac{1}{1-e^{-2D/\lambda}} \begin{pmatrix} e^{-r/\lambda} & e^{-(D-r)/\lambda} \end{pmatrix} \begin{pmatrix} e^{-(D-r)/\lambda} - e^{-(D+r)/\lambda} \\ e^{-r/\lambda} - e^{-(2D-r)/\lambda} \end{pmatrix}\\ &= \frac{1}{1-e^{-2D/\lambda}} (e^{-D/\lambda} - e^{-(D+2r)/\lambda} + e^{-D/\lambda} - e^{-(3D-2r)/\lambda})\\ &= \frac{e^{-D/\lambda}}{1-e^{-2D/\lambda}} (2 - e^{-2r/\lambda} - e^{-(2D-2r)/\lambda})\\ &= e^{-D/\lambda}(2 - e^{-2r/\lambda}) - \mathcal{O}(e^{-2(D-r)/\lambda})\end{aligned} \tag{41}$$

and thus

$$\begin{aligned}\max_{\boldsymbol{x}_{\text{p}i}, \boldsymbol{x}_{\text{q}j}} \boldsymbol{K} - \boldsymbol{K}_{\text{LCN}} = e^{-(D-2r)/\lambda}(1 - e^{-2r/\lambda}(2 - e^{-2r/\lambda})\\ + \mathcal{O}(e^{-2D/\lambda})).\end{aligned} \tag{42}$$

For pure Nyström we need to consider the distances inside a cluster. In the worst case two points overlap, i.e. $\boldsymbol{K}_{ij} = 1$, and lie at the boundary of the cluster. Since $r \ll D$ we again use Lemma A to only consider the landmark in the concerned cluster, adding an error of $\mathcal{O}(e^{-2(D-r)/\lambda})$.

$$\boldsymbol{K}_{\text{Nys},ij} = e^{-2r/\lambda} + \mathcal{O}(e^{-2(D-r)/\lambda}) \tag{43}$$

□

Note that when ignoring the effect from other clusters we can generalize the Nyström error to $l \leq d$ landmarks per cluster. In this case, because of symmetry we can optimize the worst-case distance from all cluster landmarks by putting them on an $(l-1)$-simplex centered on the cluster center. Since there are at most d landmarks in each cluster there is always one direction in which the worst-case points are r away from all landmarks. The circumradius of an $(l-1)$-simplex with side length Δ is $\sqrt{\frac{l-1}{2l}}\Delta$. Thus, the maximum distance to all landmarks is $\sqrt{r^2 + \frac{l-1}{2l}\Delta^2}$. Using Lemma B we therefore obtain the Nyström approximation

$$\boldsymbol{K}^{\text{multi}}_{\text{Nys},ij} = \frac{l e^{-2\sqrt{r^2 + \frac{l-1}{2l}\Delta^2}/\lambda}}{1 + (l-1)e^{-\Delta/\lambda}} + \mathcal{O}(e^{-2(D-r)/\lambda}) \tag{44}$$

E. Notes on Theorem 3

Lemmas C-F and and thus Theorem 1 by Altschuler et al. (2019) are also valid for $\boldsymbol{Q}$ outside the simplex so long as $\|\boldsymbol{Q}\|_1 = \sum_{i,j} |\boldsymbol{Q}_{ij}| = n$ and it only has non-negative entries. Any $\tilde{\boldsymbol{P}}$ returned by Sinkhorn fulfills these conditions if the kernel matrix is non-negative and has support. Therefore the rounding procedure given by their Algorithm 4 is not necessary for this result.

Furthermore, to be more consistent with Theorems 1 and 2 we use the L_2 distance instead of L_2^2 in this theorem, which only changes the dependence on ρ.

F. Notes on Theorem 4

To adapt Theorem 1 by Dvurechensky et al. (2018) to sparse matrices (i.e. matrices with some $\boldsymbol{K}_{ij} = 0$) we need to redefine

$$\nu := \min_{i,j} \{\boldsymbol{K}_{ij} | \boldsymbol{K}_{ij} > 0\}, \tag{45}$$

i.e. take the minimum only w.r.t. non-zero elements in their Lemma 1. We furthermore need to consider sums exclusively over these non-zero elements instead of the full $\mathbf{1}$ vector in their Lemma 1.

The Sinkhorn algorithm converges since the matrix has support (Sinkhorn & Knopp, 1967). However, the point it converges to might not exist because we only require support, not total support. Therefore, we need to consider slightly perturbed optimal vectors for the proof, i.e. define a negligibly small $\tilde{\varepsilon} \ll \varepsilon, \varepsilon'$ for which $|B(u^*, v^*)\mathbf{1} - r| \leq \tilde{\varepsilon}$, $|B(u^*, v^*)^T\mathbf{1} - c| \leq \tilde{\varepsilon}$. Support furthermore guarantees that no row or column is completely zero, thus preventing any unconstrained u_k or v_k, and any non-converging row or column sum of $B(u_k, v_k)$. With these changes in place all proofs work the same as in the dense case.

G. Proof of Proposition 1

Theorem A (Danskin's theorem). *Consider a continuous function $\phi : \mathbb{R}^k \times Z \to \mathbb{R}$, with the compact set $Z \subset \mathbb{R}^j$. If $\phi(\boldsymbol{x}, \boldsymbol{z})$ is convex in $\boldsymbol{x}$ for every $\boldsymbol{z} \in Z$ and $\phi(\boldsymbol{x}, \boldsymbol{z})$ has a unique maximizer $\bar{\boldsymbol{z}}$, the derivative of*

$$f(\boldsymbol{x}) = \max_{\boldsymbol{z} \in Z} \phi(\boldsymbol{x}, \boldsymbol{z}) \tag{46}$$

is given by the derivative at the maximizer, i.e.

$$\frac{\partial f}{\partial \boldsymbol{x}} = \frac{\partial \phi(\boldsymbol{x}, \bar{\boldsymbol{z}})}{\partial \boldsymbol{x}}. \tag{47}$$

We start by deriving the derivatives of the distances. To show that the Sinkhorn distance fulfills the conditions for Danskin's theorem we first identify $\boldsymbol{x} = \boldsymbol{C}$, $\boldsymbol{z} = \boldsymbol{P}$, and $\phi(\boldsymbol{C}, \boldsymbol{P}) = -\langle \boldsymbol{P}, \boldsymbol{C} \rangle_\text{F} + \lambda H(\boldsymbol{P})$. We next observe that the restrictions $\boldsymbol{P}\boldsymbol{1}_m = \boldsymbol{p}$ and $\boldsymbol{P}^T\boldsymbol{1}_n = \boldsymbol{q}$ define a compact, convex set for $\boldsymbol{P}$. Furthermore, ϕ is a continuous function and linear in $\boldsymbol{C}$, i.e. both convex and concave for any finite $\boldsymbol{P}$. Finally, $\phi(\boldsymbol{C}, \boldsymbol{P})$ is concave in $\boldsymbol{P}$ since $\langle \boldsymbol{P}, \boldsymbol{C} \rangle_\text{F}$ is linear and $\lambda H(\boldsymbol{P})$ is concave. Therefore the maximizer $\bar{\boldsymbol{P}}$ is unique and Danskin's theorem applies to the Sinkhorn distance. Using

$$\begin{aligned}\frac{\partial \boldsymbol{C}_{\text{Nys},ij}}{\partial \boldsymbol{U}_{kl}} &= \frac{\partial}{\partial \boldsymbol{U}_{kl}} \left(-\lambda \log(\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}) \right) \\ &= -\lambda \delta_{ik} \frac{\boldsymbol{W}_{lj}}{\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}} = -\lambda \delta_{ik} \frac{\boldsymbol{W}_{lj}}{\boldsymbol{K}_{\text{Nys},ij}},\end{aligned} \tag{48}$$

$$\begin{aligned}\frac{\partial \boldsymbol{C}_{\text{Nys},ij}}{\partial \boldsymbol{W}_{kl}} &= \frac{\partial}{\partial \boldsymbol{W}_{kl}} \left(-\lambda \log(\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}) \right) \\ &= -\lambda \delta_{jl} \frac{\boldsymbol{U}_{ik}}{\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}} = -\lambda \delta_{jl} \frac{\boldsymbol{U}_{ik}}{\boldsymbol{K}_{\text{Nys},ij}},\end{aligned} \tag{49}$$

$$\begin{aligned}\frac{\bar{\boldsymbol{P}}_{\text{Nys},ij}}{\boldsymbol{K}_{\text{Nys},ij}} &= \frac{\sum_b \bar{\boldsymbol{P}}_{U,ib} \bar{\boldsymbol{P}}_{W,bj}}{\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}} = \frac{\bar{\boldsymbol{s}}_i \bar{\boldsymbol{t}}_j \sum_b \boldsymbol{U}_{ib} \boldsymbol{W}_{bj}}{\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}} \\ &= \bar{\boldsymbol{s}}_i \bar{\boldsymbol{t}}_j \frac{\sum_b \boldsymbol{U}_{ib} \boldsymbol{W}_{bj}}{\sum_a \boldsymbol{U}_{ia} \boldsymbol{W}_{aj}} = \bar{\boldsymbol{s}}_i \bar{\boldsymbol{t}}_j\end{aligned} \tag{50}$$

and the chain rule we can calculate the derivative w.r.t. the cost matrix as

$$\frac{\partial d_c^\lambda}{\partial \boldsymbol{C}} = -\frac{\partial}{\partial \boldsymbol{C}} \left(-\langle \bar{\boldsymbol{P}}, \boldsymbol{C} \rangle_\text{F} + \lambda H(\bar{\boldsymbol{P}}) \right) = \bar{\boldsymbol{P}}, \tag{51}$$

$$\begin{aligned}\frac{\partial d_{\text{LCN},c}^\lambda}{\partial \boldsymbol{U}_{kl}} &= \sum_{i,j} \frac{\partial \boldsymbol{C}_{\text{Nys},ij}}{\partial \boldsymbol{U}_{kl}} \frac{\partial d_{\text{LCN},c}^\lambda}{\partial \boldsymbol{C}_{\text{Nys},ij}} = -\lambda \sum_{i,j} \delta_{ik} \boldsymbol{W}_{lj} \frac{\bar{\boldsymbol{P}}_{\text{Nys},ij}}{\boldsymbol{K}_{\text{Nys},ij}} \\ &= -\lambda \sum_{i,j} \delta_{ik} \boldsymbol{W}_{lj} \bar{\boldsymbol{s}}_i \bar{\boldsymbol{t}}_j = -\lambda \bar{\boldsymbol{s}}_k \sum_j \boldsymbol{W}_{lj} \bar{\boldsymbol{t}}_j \\ &= \left(-\lambda \bar{\boldsymbol{s}} (\boldsymbol{W} \bar{\boldsymbol{t}})^T \right)_{kl},\end{aligned} \tag{52}$$

$$\begin{aligned}\frac{\partial d_{\text{LCN},c}^\lambda}{\partial \boldsymbol{W}_{kl}} &= \sum_{i,j} \frac{\partial \boldsymbol{C}_{\text{Nys},ij}}{\partial \boldsymbol{W}_{kl}} \frac{\partial d_{\text{LCN},c}^\lambda}{\partial \boldsymbol{C}_{\text{Nys},ij}} = -\lambda \sum_{i,j} \delta_{jl} \boldsymbol{U}_{ik} \frac{\bar{\boldsymbol{P}}_{\text{Nys},ij}}{\boldsymbol{K}_{\text{Nys},ij}} \\ &= -\lambda \sum_{i,j} \delta_{jl} \boldsymbol{U}_{ik} \bar{\boldsymbol{s}}_i \bar{\boldsymbol{t}}_j = -\lambda \left(\sum_i \bar{\boldsymbol{s}}_i \boldsymbol{U}_{ik} \right) \bar{\boldsymbol{t}}_l \\ &= \left(-\lambda (\bar{\boldsymbol{s}}^T \boldsymbol{U})^T \bar{\boldsymbol{t}}^T \right)_{kl},\end{aligned} \tag{53}$$

and $\frac{\partial d_{\text{LCN},c}^\lambda}{\partial \log \boldsymbol{K}^\text{sp}}$ and $\frac{\partial d_{\text{LCN},c}^\lambda}{\partial \log \boldsymbol{K}_\text{Nys}^\text{sp}}$ follow directly from $\frac{\partial d_c^\lambda}{\partial \boldsymbol{C}}$. We can then backpropagate in time $\mathcal{O}((n+m)l^2)$ by computing the matrix-vector multiplications in the right order. □

H. Choosing LSH neighbors and Nyström landmarks

We focus on two LSH methods for obtaining near neighbors. Cross-polytope LSH (Andoni et al., 2015) uses a random projection matrix $\boldsymbol{R} \in \mathbb{R}^{d \times b/2}$ with the number of hash buckets b, and then decides on the hash bucket via $h(\boldsymbol{x}) = \arg\max([\boldsymbol{x}^T\boldsymbol{R} \,\|\, {-\boldsymbol{x}^T\boldsymbol{R}}])$, where $\|$ denotes concatenation. k-means LSH computes k-means and uses the clusters as hash buckets.

We further improve the sampling probabilities of cross-polytope LSH via the AND-OR construction. In this scheme we calculate $B \cdot r$ hash functions, divided into B sets (hash bands) of r hash functions each. A pair of points is considered as neighbors if any hash band matches completely. k-means LSH does not work well with the AND-OR construction since its samples are highly correlated. For large datasets we use hierarchical k-means instead (Paulevé et al., 2010; Nistér & Stewénius, 2006).

The 3D point clouds, uniform data and the graph transport network (GTN) use the L_2 distance between embeddings as a cost function. For these we use (hierarchical) k-means LSH and k-means Nyström in both sparse Sinkhorn and LCN-Sinkhorn.

Word embedding similarities are measured via a dot product. In this case we use cross-polytope LSH for sparse Sinkhorn in this case. For LCN-Sinkhorn we found that using k-means LSH works better with Nyström using k-means++ sampling than cross-polytope LSH. This is most likely due to a better alignment between LSH samples and Nyström. We convert the cosine similarity to a distance via $d_\text{cos} = \sqrt{1 - \frac{\boldsymbol{x}_\text{p}^T \boldsymbol{x}_\text{q}}{\|\boldsymbol{x}_\text{p}\|_2 \|\boldsymbol{x}_\text{q}\|_2}}$ (Berg et al., 1984) to use k-means with dot product similarity. Note that this is actually based on cosine similarity, not the dot product. Due to the balanced nature of OT we found this more sensible than maximum inner product search (MIPS). For both experiments we also experimented with uniform and recursive RLS sampling but found that the above mentioned methods work better.

I. Implementational details

Our implementation runs in batches on a GPU via PyTorch (Paszke et al., 2019) and PyTorch Scatter (Fey & Lenssen, 2019). To avoid over- and underflows we use log-stabilization throughout, i.e. we save all values in log-space and compute all matrix-vector products and additions via the log-sum-exp trick $\log \sum_i e^{x_i} = \max_j x_j + \log(\sum_i e^{x_i - \max_j x_j})$. Since the matrix $\boldsymbol{A}$ is small we compute its inverse using double precision to improve stability. Surprisingly, we did not observe any benefit from using the Cholesky decomposition or not calculating $\boldsymbol{A}^{-1}$ and instead solving the equation $\boldsymbol{B} = \boldsymbol{A}\boldsymbol{X}$ for $\boldsymbol{X}$. We furthermore precompute $\boldsymbol{W} = \boldsymbol{A}^{-1}\boldsymbol{V}$ to avoid unnecessary operations.

We use 3 layers and an embedding size $H_\text{N} = 32$ for GTN. The MLPs use a single hidden layer, biases and LeakyReLU non-linearities. The single-head MLP uses an output size of $H_\text{N, match} = H_\text{N}$ and a hidden embedding size of $4H_\text{N}$, i.e. the same as the concatenated node embedding, and the multi-head MLP uses a hidden embedding size of H_N. To stabilize initial training we scale the node embeddings by $\frac{\bar{d}}{\bar{n}\sqrt{H_\text{N, match}}}$ directly before calculating OT. $\bar{d}$ denotes the average graph distance in the training set, $\bar{n}$ the average number of nodes per graph, and $H_\text{N, match}$ the matching embedding size, i.e. 32 for single-head and 128 for multi-head OT.

For the graph datasets, the 3D point clouds and random data we use the L_2 distance for the cost function. For word embedding alignment we use the dot product, since this best resembles their generation procedure.

J. Graph dataset generation and experimental details

The dataset statistics are summarized in Table 5. Each dataset contains the distances between all graph pairs in each split, i.e. 10 296 and 1128 distances for preferential attachment. The AIDS dataset was generated by randomly sampling graphs with at most 30 nodes from the original AIDS dataset (Riesen & Bunke, 2008). Since not all node types are present in the training set and our choice of GED is permutation-invariant w.r.t. types, we permuted the node types so that there are no previously unseen types in the validation and test sets. For the preferential attachment datasets we first generated 12, 4, and 4 undirected "seed" graphs (for train, val, and test) via the initial attractiveness model with randomly chosen parameters: 1 to 5 initial nodes, initial attractiveness of 0 to 4 and $1/2\bar{n}$ and $3/2\bar{n}$ total nodes, where $\bar{n}$ is the average number of nodes (20, 200, 2000, and 20 000). We then randomly label every node (and edge) in these graphs uniformly. To obtain the remaining graphs we edit the "seed" graphs between $\bar{n}/40$ and $\bar{n}/20$ times by randomly adding, type editing, or removing nodes and edges. Editing nodes and edges is 4x and adding/deleting edges 3x as likely as adding/deleting nodes. Most of these numbers were chosen arbitrarily, aiming to achieve a somewhat reasonable dataset and process. We found that the process of first generating seed graphs and subsequently editing these is crucial for obtaining meaningfully structured data to learn from. For the GED we choose an edit cost of 1 for changing a node or edge type and 2 for adding or deleting a node or an edge.

We represent node and edge types as one-hot vectors. We train all models except SiamMPNN (which uses SGD) and GTN on Linux with the Adam optimizer and mean squared error (MSE) loss for up to 300 epochs and reduce the learning rate by a factor of 10 every 100 steps. On Linux we train for up to 1000 epochs and reduce the learning rate by a factor of 2 every 100 steps. We use the parameters from the best epoch based on the validation set. We choose hyperparameters for all models using multiple steps of grid search on the validation set, see Tables 6 to 8 for the final values. We use the originally published result of SimGNN on Linux and thus don't provide its hyperparameters. GTN uses 500 Sinkhorn iterations. We obtain the final entropy regularization parameter from λ_base via $\lambda = \lambda_\text{base} \frac{\bar{d}}{\bar{n}} \frac{1}{\log n}$, where $\bar{d}$ denotes the average graph distance and $\bar{n}$ the average number of nodes per graph in the training set. The factor $\bar{d}/\bar{n}$ serves to estimate the embedding distance scale and $1/\log n$ counteracts the entropy scaling with $n \log n$. Note that the entropy regularization parameter was small, but always far from 0, which shows that entropy regularization actually has a positive effect on learning. On the pref. att. 200 dataset we use no L_2 regularization, $\lambda_\text{base} = 0.5$, and a batch size of 200. For pref. att. 2k we use $\lambda_\text{base} = 2$ and a batch size of 20 for full Sinkhorn and 100 for LCN-Sinkhorn. For pref. att. 20k we use $\lambda_\text{base} = 50$ and a batch size of 4. λ_base scales with graph size due to normalization of the PM kernel.

For LCN-Sinkhorn we use roughly 10 neighbors for LSH (20 k-means clusters) and 10 k-means landmarks for Nyström on pref. att. 200. We double these numbers for pure Nyström Sinkhorn, sparse Sinkhorn, and multiscale OT. For pref. att. 2k we use around 15 neighbors ($10 \cdot 20$ hierarchical clusters) and 15 landmarks and for pref. att. 20k we use roughly 30 neighbors ($10 \cdot 10 \cdot 10$ hierarchical clusters) and 20 landmarks. The number of neighbors for the 20k dataset is higher and strongly varies per iteration due to the unbalanced nature of hierarchical k-means. This increase in neighbors and landmarks and PyTorch's missing support for ragged tensors largely explains LCN-Sinkhorn's deviation from perfectly linear runtime scaling.

We perform all runtime measurements on a compute node using one Nvidia GeForce GTX 1080 Ti, two Intel Xeon E5-2630 v4, and 256GB RAM.

Table 5: Graph dataset statistics.

	Graph type	Distance	Distance (test set) Mean	Distance (test set) Std. dev.	Graphs train/val/test	Avg. nodes per graph	Avg. edges per graph	Node types	Edge types
AIDS30	Molecules	GED	50.5	16.2	144/48/48	20.6	44.6	53	4
Linux	Program dependence	GED	0.567	0.181	600/200/200	7.6	6.9	7	-
Pref. att.	Initial attractiveness	GED	106.7	48.3	144/48/48	20.6	75.4	6	4
Pref. att. 200	Initial attractiveness	PM	0.400	0.102	144/48/48	199.3	938.8	6	-
Pref. att. 2k	Initial attractiveness	PM	0.359	0.163	144/48/48	2045.6	11330	6	-
Pref. att. 20k	Initial attractiveness	PM	0.363	0.151	144/48/48	20441	90412	6	-

Table 6: Hyperparameters for the Linux dataset.

	lr	batchsize	layers	emb. size	L_2 reg.	λ_{base}
SiamMPNN	1×10^{-4}	256	3	32	5×10^{-4}	-
GMN	1×10^{-4}	20	3	64	0	-
GTN, 1 head	0.01	1000	3	32	1×10^{-6}	1.0
8 OT heads	0.01	1000	3	32	1×10^{-6}	1.0
Balanced OT	0.01	1000	3	32	1×10^{-6}	2.0

Table 7: Hyperparameters for the AIDS dataset.

	lr	batchsize	layers	emb. size	L_2 reg.	λ_{base}
SiamMPNN	1×10^{-4}	256	3	32	5×10^{-4}	-
SimGNN	1×10^{-3}	1	3	32	0.01	-
GMN	1×10^{-2}	128	3	32	0	-
GTN, 1 head	0.01	100	3	32	5×10^{-3}	0.1
8 OT heads	0.01	100	3	32	5×10^{-3}	0.075
Balanced OT	0.01	100	3	32	5×10^{-3}	0.1
Nyström	0.015	100	3	32	5×10^{-3}	0.2
Multiscale	0.015	100	3	32	5×10^{-3}	0.2
Sparse OT	0.015	100	3	32	5×10^{-3}	0.2
LCN-OT	0.015	100	3	32	5×10^{-3}	0.2

Table 8: Hyperparameters for the preferential attachment GED dataset.

	lr	batchsize	layers	emb. size	L_2 reg.	λ_{base}
SiamMPNN	1×10^{-4}	256	3	64	1×10^{-3}	-
SimGNN	1×10^{-3}	4	3	32	0	-
GMN	1×10^{-4}	20	3	64	0	-
GTN, 1 head	0.01	100	3	32	5×10^{-4}	0.2
8 OT heads	0.01	100	3	32	5×10^{-3}	0.075
Balanced OT	0.01	100	3	32	5×10^{-4}	0.2
Nyström	0.02	100	3	32	5×10^{-5}	0.2
Multiscale	0.02	100	3	32	5×10^{-5}	0.2
Sparse OT	0.02	100	3	32	5×10^{-5}	0.2
LCN-OT	0.02	100	3	32	5×10^{-5}	0.2

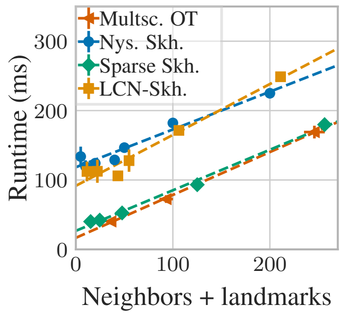

Figure 3: Runtime scales linearly with the number of neighbors/landmarks for all relevant Sinkhorn approximation methods.

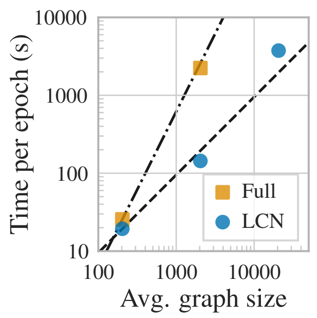

Figure 4: Log-log runtime per epoch for GTN with full Sinkhorn and LCN-Sinkhorn. LCN-Sinkhorn scales almost linearly with graph size while sustaining similar accuracy.

K. Runtimes

Table 9 compares the runtime of the full Sinkhorn distance with different approximation methods using 40 neighbors/landmarks. We separate the computation of approximate $\boldsymbol{K}$ from the optimal transport computation (Sinkhorn iterations), since the former primarily depends on the LSH and Nyström methods we choose. We observe a 2-4x speed difference between sparse (multiscale OT and sparse Sinkhorn) and low-rank approximations (Nyström Sinkhorn and LCN-Sinkhorn), while factored OT is multiple times slower due to its iterative refinement scheme. In Fig. 3 we observe that this runtime gap stays constant independent of the number of neighbors/landmarks, i.e. the relative difference decreases as we increase the number of neighbors/landmarks. This gap could either be due to details in low-level CUDA implementations and hardware or the fact that low-rank approximations require 2x as many multiplications for the same number of neighbors/landmarks. In either case, both Table 9 and Fig. 3 show that the runtimes of all approximations scale linearly both in the dataset size and the number of neighbors and landmarks, while full Sinkhorn scales quadratically.

We furthermore investigate whether GTN with approximate Sinkhorn indeed scales log-linearly with the graph size by generating preferential attachment graphs with 200, 2000, and 20 000 nodes (±50 %). We use the Pyramid matching (PM) kernel (Nikolentzos et al., 2017) as prediction target. Fig. 4 shows that the runtime of LCN-Sinkhorn scales almost linearly (dashed line) and regular full Sinkhorn quadraticly (dash-dotted line) with the number of nodes, despite both achieving similar accuracy and LCN using slightly more neighbors and landmarks on larger graphs to sustain good accuracy. Full Sinkhorn went out of memory for the largest graphs.

L. Distance approximation

Fig. 5 shows that for the chosen $\lambda = 0.05$ sparse Sinkhorn offers the best trade-off between computational budget and distance approximation, with LCN-Sinkhorn and multiscale OT coming in second. Factored OT is again multiple times slower than the other methods. Note that d_c^λ can be negative due to the entropy offset. This picture changes as we increase the regularization. For higher regularizations LCN-Sinkhorn is the most precise at constant computational budget (number of neighbors/landmarks). Note that the crossover points in this figure roughly coincide with those in Fig. 2. Keep in mind that usually the OT plan is more important than the raw distance approximation, since it determines the training gradient and tasks like embedding alignment don't use the distance at all. This becomes evident in the fact that sparse Sinkhorn achieves a better distance approximation than LCN-Sinkhorn but performs worse in both downstream tasks investigated in Sec. 8.

Table 9: Runtimes (ms) of Sinkhorn approximations for EN-DE embeddings at different dataset sizes. Full Sinkhorn scales quadratically, while all approximationes scale at most linearly with the size. Sparse approximations are 2-4x faster than low-rank approximations, and factored OT is multiple times slower due to its iterative refinement scheme. Note that similarity matrix computation time ($\boldsymbol{K}$) primarily depends on the LSH/Nyström method, not the OT approximation.

	$N = 10000$		$N = 20000$		$N = 50000$	
	$\boldsymbol{K}$	OT	$\boldsymbol{K}$	OT	$\boldsymbol{K}$	OT
Full Sinkhorn	8	2950	29	11 760	OOM	OOM
Factored OT	29	809	32	1016	55	3673
Multiscale OT	90	48	193	61	521	126
Nyström Skh.	29	135	41	281	79	683
Sparse Skh.	42	46	84	68	220	137
LCN-Sinkhorn	101	116	242	205	642	624

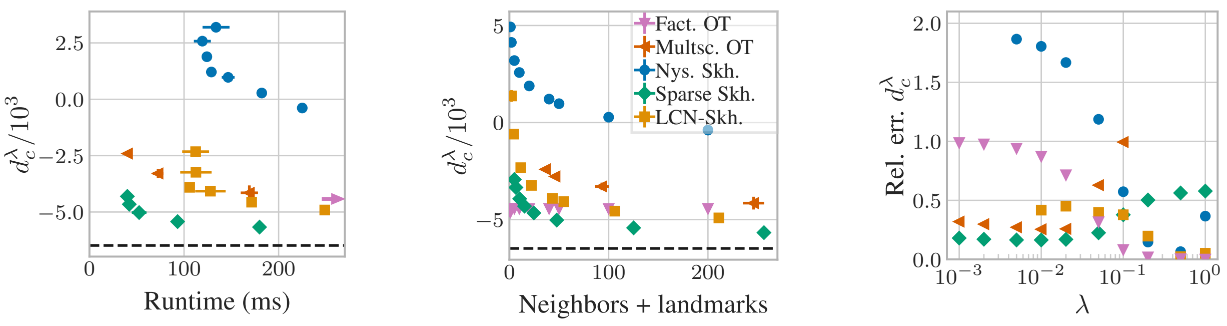

Figure 5: Sinkhorn distance approximation for different runtimes and computational budgets (both varied via the number of neighbors/landmarks), and entropy regularization parameters λ. The dashed line denotes the true Sinkhorn distance. The arrow indicates factored OT results far outside the depicted range. **Left:** Sparse Sinkhorn consistently performs best across all runtimes. **Center:** Sparse Sinkhorn mostly performs best, with LCN-Sinkhorn coming in second, and factored OT being seemingly independent from the number of neighbors. **Right:** Sparse Sinkhorn performs best for low λ, LCN-Sinkhorn for moderate and high λ and factored OT for very high λ.