Breaking the Quadratic Space Barrier for Optimal Transport via Cost Matrix Compression

Anonymous authors
Paper under double-blind review

Abstract

Optimal transport (OT) is a fundamental tool for comparing probability distributions, but the quadratic memory requirements of dense solvers limit scalability, especially on GPUs. We compress the ground-cost matrix using a sampling scheme inspired by Thorup–Zwick. We retain exact distances to nearby points and approximate the remaining distances through $\sqrt{n}$ sampled landmarks. For arbitrary ground metrics, this gives at most factor-3 distortion using $O(n^{3/2})$ expected space. More generally, a k-level scheme uses $O(kn^{1+1/k})$ expected space with distortion $2k-1$. We obtain two results using this compression scheme.

First, we adapt the push–relabel framework of Lahn, Raghvendra, and Zhang to compute an ε-additive approximation under the compressed costs using $O(n^{3/2})$ expected space, $O(\log n/\varepsilon^2)$ parallel time, and $O(n^{3/2}/\varepsilon^2)$ expected solver work. Under the original metric, the guarantee combines factor-3 distortion with ε additive error. In typical instances, especially when the input distributions are similar, much of the transport mass flows along local pairs whose costs are preserved exactly. Consequently, the compression yields high accuracy while substantially reducing storage. Our GPU implementation scales to approximately 600k points on RTX 2060, compared to approximately 15k for dense Sinkhorn implementations, with solution costs between 1.01 and 1.35 times the optimum on the evaluated high-dimensional datasets.

Second, the compressed representation yields a $(15+\varepsilon)$-approximation for incremental metric bipartite matching with $\widetilde{O}(n)$ expected amortized insertion time and $O(n^{3/2})$ expected space after $O(n^2)$ preprocessing, for fixed $\varepsilon > 0$. This improves the superlinear update-time bound of Seth et al. (ICLR, 2026) for constant-factor approximation.

1 Introduction

Optimal Transport (OT) provides a principled framework to compare probability distributions by measuring the minimum cost required to transform one into another. Given two distributions μ and ν supported on points in a metric space (X, d), the ground cost $d(a, b)$ is the cost of transporting one unit mass between support points a and b. The p-Wasserstein distance is

$$W_p(\mu,\nu) = \left(\min_{\pi \in \Pi(\mu,\nu)} \sum_{i,j} \pi_{i,j} d(a_i, b_j)^p \right)^{1/p} \tag{1}$$

where $\Pi(\mu,\nu)$ is the set of all couplings with marginals μ and ν. When both distributions consist of n uniformly weighted points, OT reduces to minimum-cost perfect matching on a complete bipartite graph. Beyond the scalar Wasserstein distance value, the resulting transport plan is useful in applications including generative modeling (4), domain adaptation (8), and point cloud registration (30).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Despite its broad applicability, the computational complexity of OT remains a major bottleneck. In the discrete setting, the problem reduces to a minimum-cost flow instance on a complete bipartite graph with n^2 edges. Classical exact algorithms require $O(n^3)$ time (31; 23), and even approximate algorithms typically require constructing and storing the full $n \times n$ cost matrix (9). This quadratic space requirement quickly becomes the dominant limitation in large-scale settings (24). To address these challenges, research has focused on two primary approximation guarantees for the W_p distance: **relative α-approximations**, which return a transport plan $\hat{M}$ whose cost yields a distance $\hat{W}$ such that $W_p \leq \hat{W} \leq \alpha W_p$ for some $\alpha > 1$, and **additive ε-approximations**, which return a transport plan $\hat{M}$ whose cost yields a distance $\hat{W}$ such that $|\hat{W} - W_p| \leq \varepsilon\Delta$, where Δ is the diameter of the metric space. Throughout we assume that $\Delta = 1$.

A major practical development was entropy-regularized OT and the corresponding Sinkhorn algorithm (9), which produces a ε-additive approximation of the optimal transport in $n^2/\varepsilon^{O(1)}$ time using $O(n^2)$ space. It replaces the original problem with a smooth optimization problem that can be solved efficiently using iterative matrix scaling. The algorithm is highly parallelizable and maps well to GPUs. Lahn, Raghvendra and Zhang (19) extended the push-relabel framework to compute an ε-additive approximation for OT on the GPU. However these methods still require explicit access to the full cost matrix, incurring $\Theta(n^2)$ memory.

Several recent works reduce the time and space complexity of OT when the transport plan has low-complexity structure. In low-rank optimal transport (LOT), the transport coupling is restricted to have non-negative rank $r \ll n$ (26; 28; 27; 14). By factorizing the transport plan into two smaller matrices, these methods achieve $O(nr)$ time and space complexity and are particularly effective when the data exhibits latent cluster structure or lies near a low-dimensional manifold (26). However, LOT leads to a non-convex optimization problem that is generally NP-hard and often requires heuristic solvers or careful initialization. Furthermore, approximation quality is sensitive to the choice of rank: small ranks may fail to capture the fine-grained ground-metric geometry, while larger ranks reduce the computational gains. In addition, low-rank OT methods often produce diffuse transport maps rather than explicit sparse transport plans or matchings. Recently, Halmos et al. (13) considered the special case of minimum-cost matching and recover an explicit matching through hierarchical refinement of a low-rank OT solution.

Other scalable approaches reduce complexity by approximating the ground metric itself. One line of work embeds the metric into simpler spaces, such as tree metrics, where OT can be computed efficiently. This yields near-linear-time algorithms with a relative $O(\log n)$ approximation guarantees for the 1-Wasserstein/EMD setting (10; 15), with extensions to higher values of p (18). More recent sampling-based metric-compression techniques, inspired by the sketches of Thorup and Zwick, combine random sampling with dynamic bichromatic closest-pair data structures to obtain scalable constant-factor approximation algorithms for OT (1; 18). Additive solvers can also be used to obtain relative approximations. Agarwal et al. (2) boost an additive solver to obtain a relative approximation for minimum-cost Euclidean bipartite matching. Seth et al. (29) use a push-relabel framework similar to that of the additive approximation algorithm of Lahn, Raghvendra, and Zhang (19) to obtain a relative approximation for incremental minimum-cost bipartite matching in an arbitrary metric. In this setting, n servers are known in advance, and each request insertion requires updating the matching of all requests received so far to distinct servers. For any fixed $\delta > 0$, their algorithm achieves an $O(\delta^{-\log_3 2})$-approximation with $\widetilde{O}(n^{1+\delta})$ amortized insertion time, which is superlinear in n for a fixed constant approximation. Other approaches, such as Chamfer distance (5; 11) and sliced Wasserstein distance (25; 6), provide simple and scalable surrogates but do not directly produce transport plans with approximation guarantees for the original OT problem.

In this paper, we make four contributions towards design of scalable and highly accurate OT algorithms.

Contribution 1: Metric Compression. Given an arbitrary ground metric (X, d), we provide a compressed representation of the cost matrix with expected size $O(kn^{1+1/k})$. This

representation preserves distances between *local pairs* exactly, while all non-local pairs are bounded within a multiplicative factor of $(2k-1)$.

Theorem 1 (Metric Compression)**.** *Given a metric space (X, d), an integer $k > 1$, and two distributions μ with support A and ν with support B of size n, we can construct a proxy distance matrix d' such that (i) The representation of d' has a size $O(kn^{1+1/k})$ in expectation, and $O(kn^{1+1/k} \log n)$ with high probability, (ii) For all $a \in A, b \in B$, the proxy distance satisfies $d(a,b) \leq d'(a,b) \leq (2k-1)d(a,b)$, and, (iii) For each $b \in B$, distances to its h nearest neighbors in A are preserved exactly (i.e., $d'(a,b) = d(a,b)$), where the expected value of h is $n^{1/k}$.*

Furthermore, this compressed representation can be computed in no more than two scans of the original cost matrix.

This compression framework extends to arbitrary costs without requiring metric structure, while retaining properties (i) and (iii). The triangle inequality, however, is needed strictly to guarantee the stretch bound in (ii).

Contribution 2: Algorithmic Efficiency. We demonstrate that this representation enables a sub-quadratic-time and sub-quadratic-space algorithm for additively approximating optimal transport under the proxy distance d'.

Theorem 2 (Additive Approximate OT under Compressed Costs)**.** *Let (X, d) be a metric space and let μ, ν be distributions supported on A, B respectively, with $|A| = |B| = n$. Given parameter $k \geq 1$, and the proxy distance d' from Theorem 1, there exists an algorithm that computes a ε-additive approximation to the optimal transport cost with respect to d' in $O\left(\frac{n^{1+1/k}}{\varepsilon^2}\right)$ time using $O(kn^{1+1/k})$ space in expectation and $O(kn^{1+1/k} \log n)$ space with high probability.*

Our approximation is a 3-relative and ε-additive approximate optimal transport plan in the worst case. However, the approximation guarantee improves when most of the optimal transport cost is contributed by local edges. In particular, if γ denotes the fraction of the optimal OT cost contributed by non-local edges, then our algorithm achieves a $(1 + 2\gamma)$ relative approximation, in addition to the ε-additive error. Thus, when the optimal transport is predominantly local, the approximation error is small.

Contribution 3: Incremental Matching. We use our compressed representation to provide improved bounds for incremental metric matching.

Theorem 3 (Incremental Metric Bipartite Matching)**.** *Let S be a fixed set of n servers in a metric space, and let at most n requests arrive one at a time. For any fixed $\varepsilon > 0$, there is a randomized data structure that can be built in $O(n^2)$ time and maintains a $(15 + \varepsilon)$-approximate minimum-cost matching of all requests received so far. It uses $O(n^{3/2})$ expected space and supports insertions in $\widetilde{O}(n)$ expected amortized time, after $O(n^2)$ preprocessing.*

In contrast to the amortized super-linear update time required by Seth et al. (29) for a constant approximation, our framework achieves an amortized near-linear update time.

Contribution 4: GPU Implementation. We provide a PyTorch/CUDA-based implementation of the proxy distance computation and the OT solver. Our method scales to $n = 650$k for $k = 3$ on a single GPU with 8GB VRAM and to $n = 1.1$M on a single GPU with 16GB VRAM, substantially improving scalability over Sinkhorn. In our experiments, it is also competitive with HiRef, a recent hierarchical-refinement method based on low-rank OT, in approximation quality while being substantially faster.

Techniques: For $k = 2$, our construction samples $\sqrt{n}$ landmarks in expectation. For each point, we store exact distances to points closer than its nearest landmark and approximate the remaining distances through that landmark. Each point has $O(\sqrt{n})$ local neighbors in expectation, so the representation stores $O(n^{3/2})$ distances in expectation. The triangle inequality ensures that non-local distances are overestimated by at most a factor of 3.

We then adapt the push-relabel additive approximation algorithm of Lahn, Raghvendra, and Zhang (LRZ) (19) to operate directly on the compressed costs and compute an ε-additive

approximate solution. A central operation in their algorithm is finding an admissible edge incident on a free vertex. We perform this search by examining local pairs explicitly and handling non-local pairs collectively through their landmarks. This avoids examining all n^2 pairs and gives $O(n^{3/2}/\varepsilon^2)$ expected solver work using $O(n^{3/2})$ expected space.

The same search strategy accelerates the incremental matching algorithm of Seth et al. (29). Their algorithm builds on the level-based approach of Agarwal and Sharathkumar (STOC, 2014), using push-relabel operations in place of Hungarian searches to handle request insertions. They choose approximately $O(\log 1/\delta)$ levels and bound the total work at each level by $\widetilde{O}(n^{2+\delta})$. With two levels, their implementation takes $\widetilde{O}(n^{5/2})$ total time, giving $\widetilde{O}(n^{3/2})$ amortized time per insertion. Using our compressed representation to implement the push-relabel operations reduces the total expected time from $\widetilde{O}(n^{5/2})$ to $\widetilde{O}(n^2)$, yielding $\widetilde{O}(n)$ expected amortized time per insertion.

Retaining a constant-factor approximation presents two challenges. First, requests are unknown when we construct the compressed representation. We therefore build the compressed matrix on the fixed server set and move each arriving request to its nearest server location, showing that this increases the optimal matching cost by at most a constant factor. Second, the compressed costs may not satisfy the triangle inequality, which the analysis of Seth et al. requires. We use compressed costs at the lower level and maintain an exact matching under the original metric at the upper level, applying the triangle inequality only to the original distances. Bounding the approximation error and accounting for the work at both levels, including reassignments between them, gives the stated $(15+\varepsilon)$-approximation and near-linear expected amortized insertion time.

2 Compressed Cost Representation

First, we describe the compressed cost representation for $k = 2$. Let (X, d) be a metric space, and let μ and ν be distributions with support A and B, respectively, where $|A| = |B| = n$. We compress the $\Theta(n^2)$-sized ground-cost matrix between A and B to a representation of expected size $O(n^{3/2})$.

Construction. Sample each point of A independently with probability $1/\sqrt{n}$ to obtain A_1. For each $b \in B$, let $a_b \in A_1$ be a closest sampled point to b, and define $w_b = d(b, a_b)$. For each $b \in B$, define $N_b = \{a \in A : d(a, b) < w_b\}$. Thus, N_b consists of all points of A that are closer to b than its nearest sampled point.

The representation stores: (i) for each $b \in B$, the pair a_b and w_b, (ii) for each $b \in B$, the set N_b together with $d(a, b)$ for all $a \in N_b$, and (iii) a matrix C storing $d(a, a')$ for all $a' \in A_1$ and $a \in A$.

Size of the representation. By standard concentration bounds, $|A_1| = O(\sqrt{n})$ with high probability, and thus the cost matrix C is an $O(\sqrt{n} \times n)$-sized matrix with a total size of $O(n^{3/2})$ with high probability. In Lemma 1 (proof in Appendix B), we show that for each $b \in B$, $|N_b| = O(\sqrt{n})$ in expectation and $O(\sqrt{n} \log n)$ with high probability.

Lemma 1. *For any $b \in B$ we have $\mathbb{E}[|N_b|] = O(\sqrt{n})$ and $\mathbb{E}\left[\sum_{b \in B} |N_b|\right] = O(n^{3/2})$. Moreover, $|N_b|$ is $O(\sqrt{n} \log n)$ with high probability, and, $\sum_{b \in B} |N_b|$ is $O(n^{3/2} \log n)$ with high probability.*

Proxy cost. We define a *proxy cost* d' based on this compressed representation representation. For any pair $(a, b) \in A \times B$, its as

$$d'(a, b) = \begin{cases} d(a, b), & \text{if } a \in N_b, \\ w_b + d(a, a_b), & \text{otherwise.} \end{cases}$$

We call an edge (a, b) local if $a \in N_b$, and non-local otherwise. For local edges, $d'(a, b) = d(a, b)$. For non-local edges, we have $d(a, b) \le d'(a, b) \le 3\, d(a, b)$.

Indeed, if $a \notin N_b$, then $d(a, b) \ge w_b$, and by the triangle inequality,

$$d'(a, b) = w_b + d(a, a_b) \le d(a, b) + d(a, a_b) \le d(a, b) + d(a, b) + d(b, a_b) \le 3\, d(a, b).$$

Finally, let π^* be an optimal plan under d, and let OPT denote its cost. Define

$$\gamma = \frac{\sum_{(a,b)\notin N_b} \pi^*(a,b)\, d(a,b)}{\mathrm{OPT}}.$$

Then $\mathrm{OT}_{d'}(\mu,\nu) \le (1+2\gamma)\,\mathrm{OPT}$, which follows by evaluating π^* under d': local edges are unchanged, while non-local edges increase by at most a factor of 3.

Construction Algorithm. The compressed representation can be constructed using a sequential scan of the cost matrix. We first sample the set $A_1 \subseteq A$. We then process the entries corresponding to $A_1 \times B$ to compute, for each $b \in B$, its nearest sampled point a_b and the value w_b. Next, we process the entries corresponding to $(A \setminus A_1) \times B$, and for each pair (a,b), we add a to N_b whenever $d(a,b) < w_b$, storing the corresponding distance. Finally, we store all distances between A_1 and A, i.e., the matrix $A_1 \times A$. Each entry of the cost matrix is accessed exactly once.

3 Additive Approximate Matching

In this section, we briefly review the LRZ algorithm for minimum-cost perfect matching. In Section 3.2, we adapt it to our compressed cost matrix, obtaining $O(n^{3/2})$ expected space and $O(n^{3/2}/\varepsilon^2)$ expected work while preserving the $O(\log n/\varepsilon^2)$ parallel time. The reduction of Lahn, Mulchandani, and Raghvendra (17) can be used to extend these matching algorithms to also solve the general optimal transport leading to Theorem 2.

3.1 The LRZ Algorithm

Let $G(A \cup B,\, A \times B)$ be a complete bipartite graph with $|A| = |B| = n$. We present the algorithm for 1-Wasserstein distance. Set $\delta = \varepsilon/3$ and round each cost down to the nearest integer multiple of δ:

$$\bar{d}(a,b) \;=\; \delta\big\lfloor d(a,b)/\delta \big\rfloor.$$

This introduces at most δ additive error to the 1-Wasserstein cost.

Dual weights and δ-feasibility. The algorithm maintains a matching $M \subseteq A \times B$ and a dual weight $y(v) \in \varepsilon\mathbb{Z}$ for every vertex $v \in A \cup B$. A pair (M, y) is *δ-feasible* if

$$y(a) + y(b) \;\le\; \bar{d}(a,b) + \delta \qquad \text{for every unmatched edge } (a,b) \notin M, \tag{2}$$

$$y(a) + y(b) \;=\; \bar{d}(a,b) \qquad \text{for every matched edge } (a,b) \in M. \tag{3}$$

The *slack* of an unmatched edge is $s(a,b) = \bar{c}(a,b) + \delta - y(a) - y(b)$. An edge with $s(a,b) = 0$ is *admissible*.

Algorithm. Initialize $M = \emptyset$, set $y(b) = \varepsilon$ for every $b \in B$, and $y(a) = 0$ for every $a \in A$. Let F_B denote the set of free vertices in B at the beginning of a phase. The algorithm proceeds while $|F_B| > \delta n$:

(i) **Construct the admissible graph.** Gather all admissible edges incident to vertices in F_B to form a subgraph G'.

(ii) **Compute a maximal matching.** Compute a maximal matching M' in G'.

(iii) **Update the matching.** If a vertex of A is matched in both M and M', remove its edge in M. Then add all edges of M' to M.

(iv) **Update the dual variables.** For each newly matched vertex a, set $y(a) \leftarrow y(a) - \delta$. For each unmatched vertex $b \in F_B$, set $y(b) \leftarrow y(b) + \delta$.

After the loop terminates, any remaining free vertices are matched arbitrarily. Since the metric space has diameter 1, this final step increases the 1-Wasserstein cost by at most δ. Combining this with the approximation error introduced by cost rounding and relaxed feasibility conditions, the total cost of the computed matching is at most $3\delta = \varepsilon$ larger than the optimum (19). Consequently, the resulting 1-Wasserstein cost is an ε-additive approximation.

Complexity. Let n_i denote the number of free vertices at the beginning of phase i. Constructing the admissible graph in phase i requires $O(n \cdot n_i)$ time. Lahn et al. (19) showed that $\sum_i n_i = O(n/\varepsilon)$, which yields a total running time of $O(n^2/\varepsilon)$.

All components of the algorithm, except for computing a maximal (greedy) matching, admit efficient parallelization. To address this bottleneck, Lahn et al. (19) adapt the parallel maximal matching algorithm of Israeli and Itai (16) to the bipartite setting. In each round, every free vertex in F_B selects a random admissible edge. Each vertex in A that receives multiple proposals then approves exactly one of its incident selected edges uniformly at random. The approved edges are added to the matching. After expected $O(\log n)$ rounds, we obtain a maximal matching. Consequently, each phase can be implemented in $O(\log n)$ parallel time, leading to an overall parallel complexity of $O(\log n/\varepsilon^2)$.

3.2 Efficient Implementation for Compressed Distances

The bottleneck in the LRZ framework is the construction of the admissible graph. In phase i, if n_i denotes the number of free vertices, the standard implementation may test each free vertex against all n vertices on the opposite side, requiring $O(n_i n)$ work. We show that the compressed representation reduces this cost to $O(n_i\sqrt{n})$ per phase. Combined with the LRZ bound $\sum_i n_i = O(n/\varepsilon)$, this gives total admissibility work $O(\sqrt{n}\sum_i n_i) = O(n^{3/2}/\varepsilon)$.

We execute LRZ using the compressed distance function d'. For each $b \in B$, the representation stores its nearest landmark $a_b \in A_1$, the distance $w_b = d(b, a_b)$, and the local neighborhood $N_b = \{a \in A : d(a, b) < d(b, a_b)\}$, together with the exact distances $d(a, b)$ for $a \in N_b$. As in LRZ, all distances are rounded to integer multiples of δ. Local pairs use the rounded distance $\overline{d}(a, b)$, while non-local pairs are represented by the rounded proxy distance[1]

$$\overline{d}'(a, b) = \overline{d}(b, a_b) + \overline{d}(a_b, a).$$

The push-relabel phases maintain δ-feasibility with respect to these rounded compressed distances.

The admissible graph need not be stored explicitly. It may contain $\Theta(n_i n)$ edges in phase i, even though the compressed representation has subquadratic size. Instead, the proposal step is implemented implicitly.

For non-local edges, every free vertex b routes through its nearest landmark a_b. Note that an edge can only be admissible if $\overline{d}(b, a_b) - y(b) = \overline{d}(a_b, a) - y(a)$. So for each landmark $a' \in A_1$ and each $a \in A$, it is sufficient to maintain the shifted dual value

$$V[a'][a] = y(a) - \overline{d}(a', a).$$

Thus, all free vertices assigned to the same landmark query the same shifted dual weights over A. For each landmark, we maintain the points of A attaining the largest shifted dual value. To test whether b has an admissible non-local edge, it suffices to query this maintained maximum for a_b; all other points have positive slack and cannot yield admissible non-local edges.

Therefore, each free vertex interacts only with its explicitly stored local neighborhood N_b and the shifted-dual summary for its landmark. Since $\mathbb{E}[|N_b|] = O(\sqrt{n})$ and the landmark query is supported by the maintained summary, the work per free vertex is $O(\sqrt{n})$ rather than $O(n)$. Hence phase i requires $O(n_i\sqrt{n})$ work without explicitly constructing the full admissible graph.

The full pseudocode, together with a detailed description of each component, is given in Algorithm 1 in Appendix A.

4 Incremental Minimum-Cost Matching

Let S be a fixed set of n servers in a metric space (X, d), and let requests arrive one at a time. For each request r, we find its nearest server $q(r)$ by scanning its n incident edges

[1]Unlike in LRZ, we take the ceil of distances so that the triangle inequality of d continues to hold

and move r to $q(r)$. This takes $O(n)$ time per insertion. Requests moved to the same location remain distinct. Let OPT and D denote the optimal matching costs before and after this modification, respectively. The total distance L by which requests are moved satisfies $L \leq \mathrm{OPT}$, and the triangle inequality gives $D \leq \mathrm{OPT} + L \leq 2\mathrm{OPT}$. Assigning each original request to the server matched to its copy adds at most L to the cost. For the rest of this section, we assume that requests are at server locations and analyze the algorithm with respect to D.

The framework of Seth et al. maintains a guess ω for the optimal cost and uses it to scale and round distances at several levels. Every request starts at the lowest level. At each level except the last, the algorithm applies push-relabel, searching for admissible edges to servers that are free or matched at the same or a higher level. A free request is promoted when its scaled dual reaches the threshold for its current level. A push may displace a previously matched request, which is then processed at its current level. At the last level, requests are matched using Hungarian searches. If the number of requests reaching a level exceeds its prescribed bound, the algorithm doubles ω and processes the current request prefix again.

Consider a two-level version of this framework. For fixed $\varepsilon > 0$, choosing the level-1 dual threshold to be $\Theta(\sqrt{n})$ gives $O(n^{3/2})$ pushes and relabels per cost estimate over n insertions, including those that displace matched requests. We allow only $O(\sqrt{n})$ requests to reach level 2 before updating the estimate. A linear scan for each admissible edge gives $\widetilde{O}(n^{3/2})$ amortized insertion time.

At level 1, we use the same local-neighborhood scans and landmark summaries as in our fast implementation of LRZ. Both algorithms maintain relaxed dual feasibility and search for admissible edges under rounded costs. To support individual dual updates in the incremental algorithm, we maintain each landmark summary in a heap. An admissible-edge search checks the stored local edges and the heap for the request's nearest landmark. Each server dual update changes one entry in each of $O(\sqrt{n})$ landmark heaps. Thus, each search or dual update takes $\widetilde{O}(\sqrt{n})$ time.

For the $m = O(\sqrt{n})$ requests at level 2, we store their exact distances to all n servers and maintain a minimum-cost matching under d. Although there may be $\Theta(n)$ candidate servers, only m are matched at level 2. For each request, maintain a heap of globally free servers keyed by distance. Since free servers have level-2 dual zero, the heap minimum gives a minimum-slack edge to a free server, which ends an augmenting path. Each visited request therefore needs to examine only the $O(m)$ matched level-2 servers and one free server. A dense implementation of Hungarian search takes $O(m^2) = O(n)$ time. Creating a distance row and heap takes $O(n)$ time on promotion; deleting a newly matched server from all heaps takes $O(m \log n) = O(n)$ time. A Hungarian search is needed when a request reaches level 2 or when a level-1 push takes a server matched at level 2. In the latter case, the displaced request remains at level 2 and must be matched again. Once a server is used at level 1, it remains there for the rest of the phase. Thus, there are $O(\sqrt{n})$ promotions and at most n transfers of servers from level 2 to level 1 per fixed estimate, giving $O(n)$ Hungarian searches in total. Consequently, the expected amortized insertion time becomes $\widetilde{O}(n)$, using $O(n^{3/2})$ expected space.

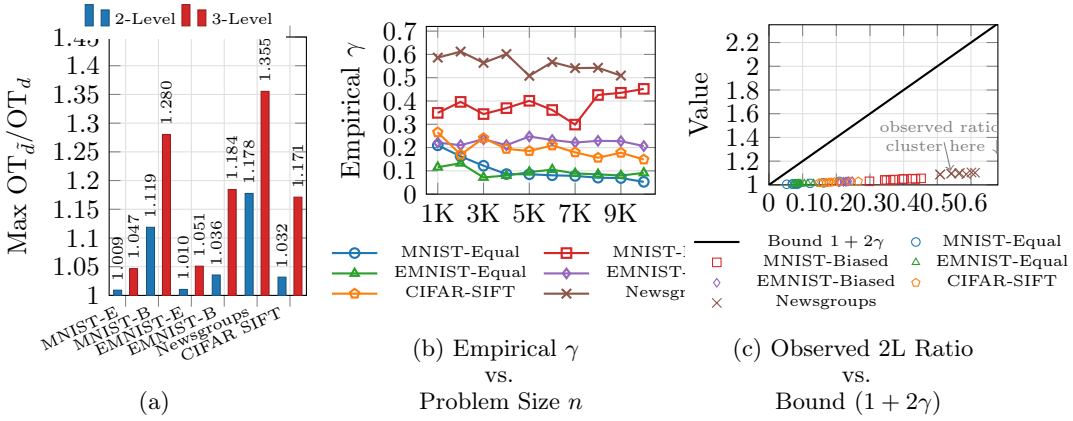
To bound the approximation, let M_1 and M_2 be the matchings at the two levels, and let $c_d(M)$ denote the cost of a matching M under d. With a suitable choice of rounding error, $c_d(M_1) \leq (3 + \varepsilon/4)D$. The triangle inequality for the original distance d implies that the remaining requests can be matched at a cost of at most $D + c_d(M_1) \leq (4 + \varepsilon/4)D$. Since M_2 is optimal for these remaining requests and servers, $c_d(M_1) + c_d(M_2) \leq (7 + \varepsilon/2)D$. Returning to the original requests gives total cost at most $L + (7 + \varepsilon/2)D \leq (15 + \varepsilon)\mathrm{OPT}$. The full analysis, including the dependence on ε, is given in the appendix.

5 Experiments

GPU Implementation and Experimental Setup: Our implementation operates directly on the compressed representation and is written using standard PyTorch tensor operations rather than custom CUDA or Triton kernels. The local neighborhoods are stored sparsely

while landmark-to-point distances and the shifted-dual summaries used by the push-relabel algorithm are maintained as dense tensors. To construct the representation without materializing the full $n \times n$ ground-cost matrix, we process pairwise distances in fixed-size tiles and reuse temporary distance buffers across tiles. Thus, the implementation explicitly controls the size of the working set, while kernel dispatch, register allocation, and shared-memory management are all handled by PyTorch and CUDA.

All experiments use an NVIDIA GeForce RTX 2060 SUPER (8 GiB) and an AMD EPYC 7302P 16-core CPU, using Python 3.11.14, PyTorch 2.11.0 with CUDA 13.0, and POT 0.9.6.post1. We evaluate on *Synthetic2D*, *20 Newsgroups* (22), *CIFAR-10 SIFT* (3), *MNIST* (20), and *EMNIST* (7). Synthetic2D uses Euclidean distance on points sampled uniformly from $[0, 1]^2$; MNIST and EMNIST use ℓ_1 distance on images; and Newsgroups and CIFAR-10 SIFT use Chamfer distance on GloVe and mean-SIFT representations (21) respectively. Since Chamfer distance is a non-metric, our multiplicative guarantee does not apply to those experiments. For MNIST and EMNIST, we consider equal sampling from the same digit classes and biased sampling from disjoint digit groups. We evaluate both 2-Level and 3-Level compression and, where feasible, compute exact OT under both the original and compressed costs.



(a) (b) Empirical γ vs. Problem Size n (c) Observed 2L Ratio vs. Bound $(1 + 2\gamma)$

Figure 1: (a) Mean distortion ratio averaged over all tested n. (b) Empirical γ validation (ε=0.01, n=1,000–10,000). (c) Observed 2-Level ratio versus the theoretical $1 + 2\gamma$ bound. Biased settings and Newsgroups have larger γ, while observed ratios remain well below the theoretical bound.

Accuracy: To isolate the representation quality from solver behavior, we compute exact OT under both the original and compressed cost matrices. Across all tested dataset and $n = 5k$-$25k$, the ratio of the OT-cost with 2-Level proxy distance and the OT-cost with true distance remains within 1.16, while the 3-Level proxy remains within 1.35 (Figure 1(a)); equal-sampling settings consistently exhibit lower distortion than biased ones. Figure 1(b) evaluates the empirical behavior of the parameter γ, the fraction of optimal transport

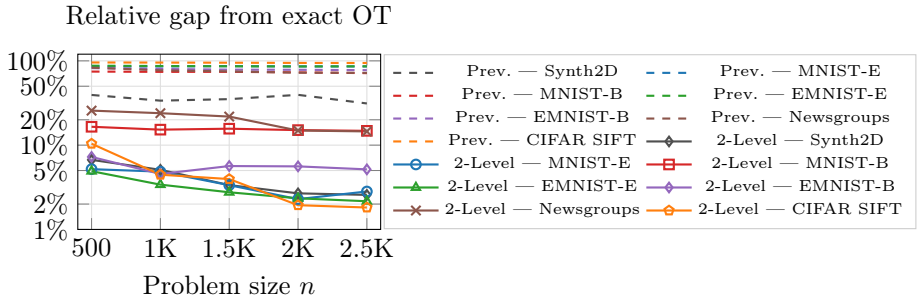


Figure 2: Comparison with (1)

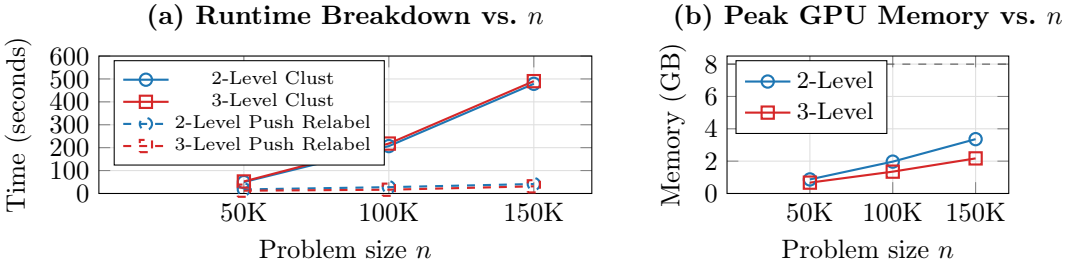


Figure 3: Runtime and Memory comparison on large EMNIST instances

cost contributed by non-local edges. Biased instances and Newsgroups exhibit larger γ. Nevertheless, plans computed under the compressed cost remain within 1.11 times the optimal when evaluated using the original ground costs (Figure 1(c)), suggesting that near-optimal matchings may use substantially fewer non-local edges than the exact optimal. We also compare against the sampling-based proxy of Agarwal et al. (1). Across all datasets and all tested values of $n \in \{500, \ldots, 2500\}$, the 2-Level proxy consistently produces transport costs substantially closer to the true optimum (Figure 2). On MNIST and EMNIST under equal sampling, the distance from (1) incurs roughly 1.86 times the true cost, while the 2-Level proxy remains below 1.05 times the true cost; under biased sampling, the relative error decreases from roughly 73% to 15%. Note that Figure 2 compares approximation quality rather than runtime; the available implementation of (1) is a CPU-based Python prototype that does not scale to our large GPU instances. In the appendix D, we provide additional details on the impact of sampling probability and accuracy of our compressed representation.

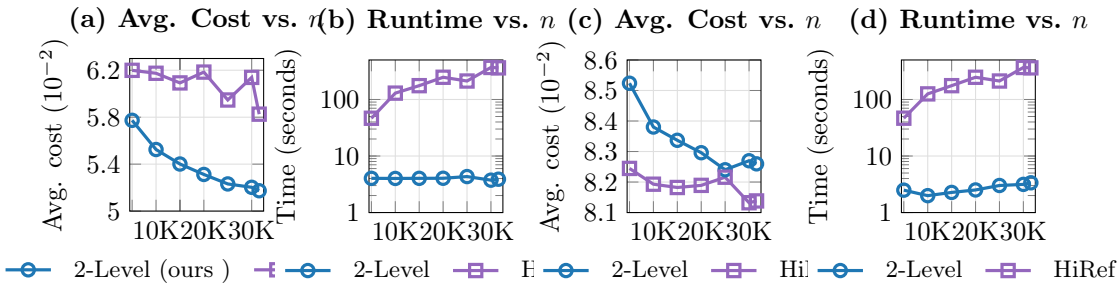


Figure 4: $(a-b)$: Comparison with HiRef on MNIST equal sampling. (a): average transport cost vs. n. (b): runtime vs. n on a log scale. 2-Level attains roughly 12× to 99× speedup over HiRef. $(c-d)$: Comparison with HiRef on MNIST biased sampling. (c): average transport cost vs. n. (d): runtime vs. n on a log scale. 2 Level attains roughly 19× to 117× speedup.

Scalability: We evaluate scalability through three experiments: maximum problem size, GPU memory usage relative to Sinkhorn, and comparison with HiRef (13). On Synthetic2D, the 3-Level solver completes instances up to $n = 650{,}000$ on a 8 GB RTX 2060 SUPER, while the 2-Level solver reaches $n = 200{,}000$. At every completed size, the push-relabel algorithm takes under 4 seconds; construction of the compressed representation dominates the runtime. We compare the 2-Level solver with HiRef on MNIST under equal and biased sampling regimes (Figure 4). Since both methods output hard matchings, the resulting costs are directly comparable. Under equal sampling, the 2-Level solver is both faster and more accurate. Under biased sampling, HiRef attains slightly lower cost, but the difference is small $(0.3\% - 3.3\%)$, while the 2-Level solver remains one to two orders of magnitude faster. Detailed values are reported in Appendix D.1.1. To evaluate scalability on real high-dimensional data, we run both solvers on EMNIST histogram images under ℓ_1 distance. Exact OT is infeasible at these scales; we report only solver runtime and peak GPU memory. Figure 3 shows that the 3-Level solver reaches larger n before exhausting memory. The compression time dominates the running time, whereas the push-relabel phase remains a small fraction of the total time.

6 AI Use Disclosure

In this work, we used generative AI tools to implement methods. We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, clean and reformat datasets, support qualitative and thematic data analysis, interpret results, design or providing feedback on research methodology or experiments, and assistance with translation is not applicable to this work. Additionally, we used generative AI tools to assist in creating the software code and edit the research paper to improve readability. We have reviewed all AI-assisted work. The correctness of the code was evaluated by the authors and it also passes feasibility tests which guarantee the optimality of the computed solutions. Each AI-generated suggestion for readability was individually evaluated by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

7 Reproducibility

In effort to ensure reproducibility of this work, we are including the source code in the supplemental materials. Within this source code, a README file is provided with the instructions needed to produce the experimental results within the paper. After the review period, a repository containing the source code will be made publicly available.

References

[1] Pankaj Agarwal, Oliver Chubet, Sharath Raghvendra, and Keegan Yao. A scalable constant-factor approximation algorithm for w_p optimal transport. In *The Fourteenth International Conference on Learning Representations*, 2026.

[2] Pankaj K Agarwal, Sharath Raghvendra, Pouyan Shirzadian, and Rachita Sowle. A higher precision algorithm for computing the 1-wasserstein distance. In *The Eleventh International Conference on Learning Representations*, 2023.

[3] Krizhevsky Alex. Learning multiple layers of features from tiny images. *https://www. cs. toronto. edu/kriz/learning-features-2009-TR. pdf*, 2009.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. Pmlr, 2017.

[5] Ainesh Bakshi, Piotr Indyk, Rajesh Jayaram, Sandeep Silwal, and Erik Waingarten. Near-linear time algorithm for the chamfer distance. *Advances in Neural Information Processing Systems*, 36:66833–66844, 2023.

[6] Nicolas Bonneel, Julien Rabin, Gabriel Peyré, and Hanspeter Pfister. Sliced and radon wasserstein barycenters of measures. *Journal of Mathematical Imaging and Vision*, 51(1):22–45, 2015.

[7] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

[8] Nicolas Courty, Rémi Flamary, Devis Tuia, and Alain Rakotomamonjy. Optimal transport for domain adaptation. *IEEE transactions on pattern analysis and machine intelligence*, 39(9):1853–1865, 2016.

[9] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

[10] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. Approximating metrics by tree metrics. *ACM SIGACT News*, 35(2):60–70, 2004.

[11] Ying Feng and Piotr Indyk. Even faster algorithm for the chamfer distance. *arXiv preprint arXiv:2505.08957*, 2025.

[12] Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, et al. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

[13] Peter Halmos, Julian Gold, Xinhao Liu, and Benjamin J Raphael. Hierarchical refinement: Optimal transport to infinity and beyond. *arXiv preprint arXiv:2503.03025*, 2025.

[14] Peter Halmos, Xinhao Liu, Julian Gold, and Benjamin J Raphael. Low-rank optimal transport through factor relaxation with latent coupling. *Advances in Neural Information Processing Systems*, 37:114374–114433, 2024.

[15] Piotr Indyk and Nitin Thaper. Fast image retrieval via embeddings. In *3rd international workshop on statistical and computational theories of vision*, volume 1. Nice, France, 2003.

[16] Amos Israeli and Alon Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):77–80, 1986.

[17] Nathaniel Lahn, Deepika Mulchandani, and Sharath Raghvendra. A graph theoretic additive approximation of optimal transport. *Advances in Neural Information Processing Systems*, 32, 2019.

[18] Nathaniel Lahn, Sharath Raghvendra, Emma Saarinen, and Pouyan Shirzadian. Scalable approximation algorithms for p-wasserstein distance and its variants. In *Forty-second International Conference on Machine Learning*, 2025.

[19] Nathaniel Lahn, Sharath Raghvendra, and Kaiyi Zhang. A combinatorial algorithm for approximating the optimal transport in the parallel and MPC settings. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[20] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[21] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[22] Tom Mitchell. Twenty newsgroups, 1997.

[23] James B Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78(2):109–129, 1997.

[24] Gabriel Peyré and Marco Cuturi. *Computational optimal transport: With applications to data science*. Now Foundations and Trends, 2019.

[25] Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot. Wasserstein barycenter and its application to texture mixing. In *International conference on scale space and variational methods in computer vision*, pages 435–446. Springer, 2011.

[26] Meyer Scetbon, Marco Cuturi, and Gabriel Peyré. Low-rank sinkhorn factorization. In *International Conference on Machine Learning*, pages 9344–9354. PMLR, 2021.

[27] Meyer Scetbon, Michal Klein, Giovanni Palla, and Marco Cuturi. Unbalanced low-rank optimal transport solvers. *Advances in Neural Information Processing Systems*, 36:52312–52325, 2023.

[28] Meyer Scetbon, Gabriel Peyré, and Marco Cuturi. Linear-time gromov wasserstein distances using low rank couplings and costs. In *International Conference on Machine Learning*, pages 19347–19365. PMLR, 2022.

[29] Ritesh Seth, Mrinal Garg, Sujoy Bhore, Sharath Raghvendra, and Syamantak Das. Efficient algorithms for incremental metric bipartite matching. In *International Conference on Learning Representations*, volume 2026, pages 57105–57130, 2026.

[30] Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics (ToG)*, 34(4):1–11, 2015.

[31] Robert E Tarjan. Dynamic trees as search trees via euler tours, applied to the network simplex algorithm. *Mathematical Programming*, 78(2):169–177, 1997.

A Pseudocode and Implementation Details for CompressedPushRelabel

Algorithm 1 gives the full pseudocode for the compressed push-relabel solver. We describe each component in detail below.

Input structures. The algorithm consumes the compressed representation built from Section 2. Concretely, for each $b \in B$ the representation stores:

- the nearest landmark $a_b \in A_1$ and the rounded threshold $\bar{w}_b = \bar{d}(b, a_b)$,
- the local neighborhood $N_b = \{a \in A : d(a, b) < w_b\}$ in CSR format, with rounded exact distances $\bar{d}(a, b)$ for each $a \in N_b$.

For the non-local proxy routes, the DR matrix stores the rounded landmark-to-red distances $\bar{d}(a', a)$ for all $a' \in A_1$, $a \in A$.

The V matrix. The matrix $V \in \mathbb{R}^{|A_1| \times n}$, defined entry-wise as

$$V[a'][a] \;=\; y(a) \;-\; \bar{d}(a',\, a),$$

encodes the dual weight of each red point a relative to each landmark's distance to a. It is initialized to $V[a'][a] = -\bar{d}(a', a)$ since $y(a) = 0$ at startup. When a matched red point a has its dual decremented by δ, the single column update $V[\,\cdot\,, a] \leftarrow V[\,\cdot\,, a] - \delta$ propagates the change across every proxy route through a in $O(|A_1|)$ time, avoiding a full matrix rebuild.

Admissibility and the two candidate sets. An edge (b, a) is δ-admissible when $y(a) + y(b) = \bar{d}'(b, a) + \delta$. The algorithm maintains two disjoint candidate sets per free vertex b:

- $\mathsf{Cands}_1(s, \tau)$ — *non-local* candidates routed via landmark a_b. The admissibility condition reduces to $V[a_b][a] = \tau$ where $\tau = \bar{w}_b + \delta - y(b)$. Crucially, all free blues sharing the same nearest landmark a_b and the same value of τ share identical Cands_1. The number of distinct (a_b, τ) pairs across all free vertices is at most $|A_1| = O(\sqrt{n})$, so the total Set 1 scan cost per phase is $O(n^{3/2})$ regardless of $|F_B|$.
- $\mathsf{Cands}_2(b)$ — *local* candidates stored explicitly in the CSR adjacency list of b. The admissibility condition is $y(a) + y(b) = \bar{d}(a, b) + \delta$, checked directly against each $a \in N_b$. The total Set 2 scan cost per phase is $O(n^{3/2})$ because the CSR lists have that total expected size.

Proposal and acceptance. Each free vertex b with at least one candidate proposes to a single candidate, choosing from Cands_1 with probability proportional to $|\mathsf{Cands}_1|$ and from Cands_2 otherwise. Each red vertex a that receives multiple proposals accepts one uniformly at random. This is the bipartite adaptation of the Israeli–Itai parallel maximal matching algorithm (16), which guarantees a maximal matching is reached within $O(\log n)$ rounds in expectation.

Algorithm 1 CompressedPushRelabel(A, B, ε)

Require: Compressed representation of (A, B) from Section 2: landmarks A_1, thresholds $\{\bar{w}_b\}_{b\in B}$, rounded DR matrix $\bar{d}(a', a)$ for $a' \in A_1$, CSR adjacency $\{N_b\}_{b\in B}$ with rounded local distances $\bar{d}(a, b)$, matrix V; parameter $\varepsilon > 0$.
Ensure: ε-additive approximate min-cost perfect matching M under the proxy distances $\bar{d}'$.

Set $\delta \leftarrow \varepsilon/3$.
$y(a) \leftarrow 0$ **for all** $a \in A$; $y(b) \leftarrow \delta$ **for all** $b \in B$; $M \leftarrow \emptyset$; $F_B \leftarrow B$.
while $|F_B| > \delta n$ **do**
  $t[b] \leftarrow \bar{w}_b + \delta - y(b)$ **for each** $b \in F_B$. ▷ admissibility threshold for non-local edges via a_b
  for all unique pairs (s, τ) in $\{(a_b, t[b]) : b \in F_B\}$ **do**
    $\mathsf{Cands}_1(s, \tau) \leftarrow \{a \in A : V[s][a] = \tau\}$. ▷ one V-row scan per pair
  end for
  for all $b \in F_B$ **do**
    $\mathsf{Cands}_2(b) \leftarrow \{a \in N_b : y(a) + y(b) = \bar{d}(a, b) + \delta\}$. ▷ δ-admissible local edges
  end for
  if $\mathsf{Cands}_1(a_b, t[b]) \cup \mathsf{Cands}_2(b) = \emptyset$ **for all** $b \in F_B$ **then**
    $\sigma \leftarrow \min_{b\in F_B}$ (minimum positive slack over all edges incident to F_B).
    $y(b) \leftarrow y(b) + \sigma$ **for all** $b \in F_B$. **Continue.**
  end if
  Each $b \in F_B$ with candidates proposes to one: Cands_1 with probability $\frac{|\mathsf{Cands}_1|}{|\mathsf{Cands}_1|+|\mathsf{Cands}_2|}$, Cands_2 otherwise.
  Each $a \in A$ with at least one proposal accepts one uniformly at random.
  Update M: add accepted pairs; return evicted blues to F_B.
  for all newly matched a **do**
    $y(a) \leftarrow y(a) - \delta$; $V[\cdot, a] \leftarrow V[\cdot, a] - \delta$. ▷ $O(|A_1|)$ column update
  end for
  $y(b) \leftarrow y(b) + \delta$ **for each** $b \in F_B$.
end while
Match any remaining free blues to free reds arbitrarily.
return M.

Dual updates. When no admissible edges exist for any free blue vertex, all free blue duals are raised by the minimum positive slack σ over all edges incident to F_B. This is the dual-ascent step that makes progress toward creating new admissible edges without violating δ-feasibility.

Termination and approximation quality. The loop terminates when $|F_B| \le \delta n$; the remaining unmatched vertices are paired arbitrarily. By the analysis of Lahn et al. (19), the total number of phases is $O(1/\varepsilon^2)$ and $\sum_i n_i = O(n/\varepsilon)$, giving $O(n^{3/2}/\varepsilon)$ total work. Since $\tilde{d}(a, b) \ge d(a, b)$ for all pairs, δ-feasibility under $\bar{d}'$ implies standard dual feasibility under d, so the output matching is an ε-additive approximation under the true ground metric. Combined with the $(1 + 2\gamma)$ multiplicative bound from Section 2, the overall approximation guarantee follows.

B Omitted Proofs

Proof of Lemma 1

Proof. Fix $b \in B$ and order the points of A as $a_1, a_2, \ldots, a_n$ such that

$$d(a_1, b) \le d(a_2, b) \le \cdots \le d(a_n, b).$$

Each point of A is sampled independently with probability $1/\sqrt{n}$. Let X denote the index of the first sampled point in this ordering. Then X is a geometric random variable with

success probability $1/\sqrt{n}$, and hence $\mathbb{E}[X] = \Theta(\sqrt{n})$. Moreover,

$$\Pr[X > 2\sqrt{n}\log n] = (1 - n^{-1/2})^{2\sqrt{n}\log n} \leq e^{-2\log n} = n^{-2}.$$

Therefore X is $O(\sqrt{n}\log n)$ with high probability.

By definition, N_b consists of the points that appear before the first sampled point, i.e., $|N_b| = X - 1$. Therefore, $\mathbb{E}[|N_b|] = \Theta(\sqrt{n})$. Summing over all $b \in B$, we obtain $\mathbb{E}\left[\sum_{b\in B} |N_b|\right] = O(n^{3/2})$. □

C Incremental Minimum-Cost Matching: Algorithm and Analysis

Let S be a fixed set of n servers in a metric space (X, d), and let $r_1, \ldots, r_n$ be a sequence of requests. After each insertion, we maintain a matching of all requests received so far to distinct servers. Assignments of previously inserted requests may change. We assume that each distance can be evaluated in constant time. We show that, for any $0 < \varepsilon \leq 1$, a $(15 + \varepsilon)$-approximate matching can be maintained in $\widetilde{O}(n/\varepsilon)$ expected amortized time per insertion, using $O(n^{3/2})$ expected space and $O(n^2)$ preprocessing.

Moving requests to their nearest servers. For each arriving request r, scan its n incident edges to find a nearest server $q(r) \in S$. Replace r by a distinct copy located at $q(r)$. This requires $O(n)$ time per insertion. Fix a request prefix $R_t = \{r_1, \ldots, r_t\}$, and let $\overline{R}_t$ be the collection of request copies. Write OPT for the optimal cost on R_t and D for the optimal cost on $\overline{R}_t$. Let

$$L = \sum_{r\in R_t} d(r, q(r)).$$

Since $q(r)$ is a nearest server, $L \leq \mathrm{OPT}$. For any fixed assignment M, let $\overline{M}$ assign each copy to the same server as its original request. The triangle inequality gives

$$|c_d(M) - c_d(\overline{M})| \leq L.$$

Applying this inequality to an optimal original matching shows that

$$D \leq \mathrm{OPT} + L \leq 2\mathrm{OPT}. \tag{1}$$

Conversely, a matching of the copies gives a matching of the original requests whose cost increases by at most L. For the remainder of the algorithmic description, all requests are assumed to be at server locations. Multiple requests at a location remain distinct vertices.

Connection with the LRZ implementation. We reuse the compressed admissible-edge search developed for LRZ; its implementation for this setting is described below. Both LRZ and the incremental push-relabel framework maintain tight matched edges and allow a rounding-scale slack on unmatched edges. After scaling, these are the unit-slack invariants stated below. At level 1, the incremental algorithm uses the same local-neighborhood scans and landmark summaries, with heaps supporting individual server dual changes. The two-level framework determines the cost estimate, caps request duals, and promotes the remaining requests to an exact matching at level 2. Thus the search implementation carries over, while the promotion and cost-estimation rules provide the incremental relative-approximation guarantee.

The two-level algorithm of Seth et al. We use a two-level specialization of the push-relabel framework of Seth et al. (29). Let M_1 and M_2 be the matchings at levels 1 and 2. At level 1, maintain a dual $y(v)$ for each request and server. For an integer edge-cost function c_λ, the dual invariants are

$$\begin{aligned} y(r) + y(s) &= c_\lambda(r, s), && (r, s) \in M_1, \\ y(r) + y(s) &\leq c_\lambda(r, s) + 1, && (r, s) \notin M_1. \end{aligned} \tag{2}$$

A non-matching edge is admissible when equality holds in the second inequality. Server duals are nonpositive. Every server not used by M_1 has level-1 dual zero.

Let $\omega > 0$ be the current estimate of D, and set

$$\lambda = \frac{\varepsilon\omega}{24n}, \qquad H = \left\lceil (72/\varepsilon + 3)\sqrt{n} \right\rceil . \tag{3}$$

The same parameters will be used for the algorithm with proxy distances. We first describe the algorithm with $c_\lambda(r,s) = \lceil d(r,s)/\lambda \rceil$. A period during which ω remains fixed is called a phase. The last part of the proof describes how to maintain $\omega \le 2D$ and at most $\lceil \sqrt{n} \rceil$ requests at level 2.

A new request starts at level 1 with dual zero. For a free request r, find an admissible edge (r,s). If one exists, match r to s and decrease $y(s)$ by one. If s was matched, its former request becomes free. Continue processing that request at its current level. If no admissible edge exists, increase $y(r)$ by the minimum slack, stopping at H. A free request whose dual reaches H moves to level 2 and retains level-1 dual H. At level 2, maintain an exact minimum-cost matching under d using servers not used by M_1.

Requests can move only from level 1 to level 2. A server used by M_1 remains at level 1, although its assigned request may change. When a level-1 push uses a server matched in M_2, its former request is matched again at level 2. These rules preserve equation (2).

Over a phase with t requests, the total increase in request duals is at most tH. Every relabel increases a request dual by at least one. Each push decreases one server dual by one, and tightness of the edges in M_1, together with nonnegative costs, gives

$$\sum_{s \text{ matched in } M_1} -y(s) \le \sum_{r \text{ matched in } M_1} y(r) \le tH. \tag{4}$$

Since these server duals never reset within a phase, there are at most tH pushes and at most tH relabels. A direct admissible-edge or minimum-slack search scans all n servers. It therefore takes $O(tn^{3/2}/\varepsilon)$ time to process the phase at level 1. Level 2 takes $\widetilde{O}(tn)$ time, as shown below. For fixed ε, this gives the $\widetilde{O}(n^{3/2})$ amortized insertion bound before using the compressed representation.

Using proxy distances at level 1. Construct the two-level distance representation on S. Specifically, choose a uniform landmark subset $A_1 \subseteq S$ of size $\lceil \sqrt{n} \rceil$. For each $q \in S$, let a_q be its nearest landmark, let $w_q = d(q, a_q)$, and let $N_q = \{s \in S : d(q,s) < w_q\}$. Store all local distances and all distances between A_1 and S. Then

$$d'(q,s) = \begin{cases} d(q,s), & s \in N_q, \\ w_q + d(a_q, s), & s \notin N_q \end{cases} \tag{5}$$

This satisfies $d(q,s) \le d'(q,s) \le 3d(q,s)$. The expected representation size is $O(n^{3/2})$ and

$$\mathbb{E}[\max_{q \in S} |N_q|] = O(\sqrt{n} \log n).$$

Construction takes $O(n^2)$ time and does not require storing the full distance matrix.

We now use d' at level 1. Round the two distances in each non-local proxy cost separately:

$$c_\lambda(q,s) = \begin{cases} \lceil d(q,s)/\lambda \rceil, & s \in N_q, \\ \lceil w_q/\lambda \rceil + \lceil d(a_q,s)/\lambda \rceil, & s \notin N_q. \end{cases} \tag{6}$$

We also write $c_\lambda(M)$ for the sum of these integer costs over a matching M. For each pair,

$$d'(q,s) \le \lambda c_\lambda(q,s) \le d'(q,s) + 2\lambda. \tag{7}$$

The dual updates and the bound in equation (4) require only nonnegative integer costs. Thus they remain valid even though d' need not be a metric.

The following heap implements the landmark summary used in the LRZ implementation: minimizing rounded distance minus server dual is equivalent to maximizing the corresponding

shifted dual value. For each landmark a, maintain a heap over the servers with keys $\lceil d(a,s)/\lambda \rceil - y(s)$. For a request at q, the minimum cost minus server dual is

$$\begin{aligned} &\min_s \{c_\lambda(q,s) - y(s)\} \\ &\quad = \min \left\{ \begin{array}{l} \min\limits_{s \in N_q} \{\lceil d(q,s)/\lambda \rceil - y(s)\}, \\ \lceil w_q/\lambda \rceil + \min\limits_{s \in S} \{\lceil d(a_q,s)/\lambda \rceil - y(s)\} \end{array} \right\}. \end{aligned} \tag{8}$$

The landmark heap includes local servers. This does not change the minimum: the triangle inequality for d implies that a rounded direct distance is at most the sum of the two rounded distances through a landmark. Subtracting $y(r)$ and adding one to equation (8) gives the minimum slack for a free request. A zero slack identifies an admissible edge; otherwise, the slack determines the next relabel.

Each query scans N_q and reads one heap minimum. A server dual change updates one entry in each of $|A_1| = O(\sqrt{n})$ heaps, costing $O(\sqrt{n} \log n)$ time. Hence the $O(tH)$ level-1 operations take $\widetilde{O}(tn/\varepsilon)$ expected time. The maximum-neighborhood bound applies to all locations in the fixed set S, so this accounting also applies when queries depend on previous matching decisions.

Number of requests at level 2. Let m be the number of requests that have reached level 2, and let M^* be an optimal matching of the modified requests, of cost D. The sum of the level-1 duals is $c_\lambda(M_1) + Hm$ after a request has been processed. Feasibility and nonpositive server duals imply

$$c_\lambda(M_1) + Hm \le c_\lambda(M^*) + t \le \frac{3D}{\lambda} + 3n. \tag{9}$$

The same upper bound applies during an insertion; a free level-1 request contributes an additional nonnegative term to the dual sum. If $\omega \ge D$, equations (3) and (9) give $m \le \sqrt{n}$. We update ω whenever m exceeds $\lceil \sqrt{n} \rceil$.

Exact matching at level 2. For each request reaching level 2, compute and store its exact distances to all n servers. Maintain a heap containing the free servers, keyed by their distances to this request. Here a server is free if it is unmatched in both M_1 and M_2. With $m = O(\sqrt{n})$ requests at level 2, the distance rows and heaps use $O(nm) = O(n^{3/2})$ space. Each row and heap is initialized in $O(n)$ time. When a server first becomes matched, delete it from all current free-server heaps, taking $O(m \log n)$ time. Once matched, a server never becomes free again during the phase.

The level-2 matching must be updated when a request first reaches this level or when a level-1 push uses a server currently matched in M_2. Each event leaves one unmatched request at level 2. A Hungarian search finds an augmenting path and restores an optimal matching under d. Removing the server used at level 1 preserves dual feasibility and tightness on the remaining level-2 matching. For a new request, choose an initially feasible level-2 dual.

There may be $\Theta(n)$ candidate servers even though $m = O(\sqrt{n})$. Scanning all of them for each visited request would take $O(mn)$ time. We avoid this scan using the maintained free-server heaps. Write $u(r)$ and $v(s)$ for the level-2 duals. All globally free servers have $v(s) = 0$, so the minimum slack from a visited request r to a free server is

$$\min_{s \text{ free}} \{d(r,s) - u(r) - v(s)\} = \min_{s \text{ free}} d(r,s) - u(r).$$

The heap supplies a minimizing server in $O(1)$ time. A free server ends an augmenting path, so it suffices to consider this one free-server edge for each visited request. All other relevant servers are matched in M_2, and there are at most m of them.

The residual search can therefore represent all free servers by a single terminal vertex, remembering which free server realizes each request's terminal edge. Besides this terminal, it has at most m requests and m matched servers. Forward edges have nonnegative slack; reverse matched edges have zero reduced cost. A dense implementation of Dijkstra's algorithm, using linear scans to select the next vertex, takes $O(m^2 + m)$ time. The usual Hungarian

dual update and augmentation then restore an optimal matching. Free servers retain dual zero: the search stops at the first free server, whose dual update is zero.

A request can be displaced repeatedly at level 2. Each such displacement is caused by a distinct transfer of a server from level 2 to level 1: once a server is used at level 1, it remains there until the cost estimate changes. Over a phase containing t requests, there are at most t promotions and at most t such server transfers, giving at most $2t$ Hungarian searches. Only $O(\sqrt{n})$ distinct requests can reach level 2 before the estimate is increased.

Building a distance row and heap costs $O(n)$ on promotion. Each newly matched free server is deleted from all current heaps in $O(m \log n)$ time, and at most t servers become matched during the phase. Since $m = O(\sqrt{n})$, a level-2 repair, including heap maintenance and the initialization of a newly promoted request when needed, costs

$$O(n + m^2 + m \log n) = O(n).$$

Thus all level-2 work in the phase takes $O(tn)$ time and uses $O(n^{3/2})$ space.

Approximation guarantee. Equation (9), together with $\omega \le 2D$, gives

$$c_d(M_1) \le \lambda c_\lambda(M_1) \le 3D + 3n\lambda \le (3 + \varepsilon/4)D. \tag{10}$$

The alternating paths in $M_1 \oplus M^*$ connect the requests at level 2 to distinct servers outside M_1. Their total length under d is at most $D + c_d(M_1)$. Matching their endpoints gives a feasible matching for level 2, and the triangle inequality for d bounds its cost by the same quantity. Since M_2 is optimal under d,

$$c_d(M_2) \le D + c_d(M_1) \le (4 + \varepsilon/4)D. \tag{11}$$

Consequently, the total cost for the modified requests is

$$c_d(M_1) + c_d(M_2) \le (7 + \varepsilon/2)D. \tag{12}$$

All matching-path arguments use the original metric d; they do not assume that d' satisfies the triangle inequality. Finally, assign each original request to the server assigned to its copy. By equation (1), its total cost is at most

$$\begin{aligned} L + (7 + \varepsilon/2)D &\le (7 + \varepsilon/2)\mathrm{OPT} + (8 + \varepsilon/2)L \\ &\le (15 + \varepsilon)\mathrm{OPT}. \end{aligned}$$

If requests already lie at server locations, $L = 0$ and $D = \mathrm{OPT}$, giving a $7 + \varepsilon$ approximation directly.

Updating the estimate and total complexity. As long as $D = 0$, each request copy can be matched to a free server at its location. When D first becomes positive, initialize ω to the minimum positive distance between server locations. If more than $\lceil \sqrt{n} \rceil$ requests reach level 2, double ω and process the current request prefix again. Equation (9) shows that an update can occur only when $\omega < D$. The new estimate is therefore less than $2D$. Since D is nondecreasing, the invariant $\omega \le 2D$ holds throughout each phase.

Let Δ_S be the ratio of the largest to the smallest positive distance between server locations. There are $O(\log(n\Delta_S))$ phases. If all servers have the same location, all modified requests can be matched at zero cost and this case is handled directly.

The compressed representation is retained between phases. To begin a new phase, reset the duals of the at most t used servers, mark servers free, discard the level-2 distance rows and heaps, and process the t requests again. Resetting the landmark heaps costs $\widetilde{O}(t\sqrt{n})$. Once all server duals are zero, doubling λ maps each old rounded distance key k to $\lceil k/2 \rceil$, which preserves heap order. Keys are evaluated from the stored distances and the current λ, so all heap entries need not be rewritten. Constructing the level-2 rows again costs at most $O(tn)$, and the previously computed nearest server of each original request is retained.

Thus a phase takes $O(tn \log n/\varepsilon)$ expected time. There are $O(\log(n\Delta_S))$ phases, and a prefix of t requests is processed at most once per phase. Consequently, the expected amortized insertion time is

$$O\left(\frac{n \log n}{\varepsilon} \log(n\Delta_S)\right) = \widetilde{O}(n/\varepsilon),$$

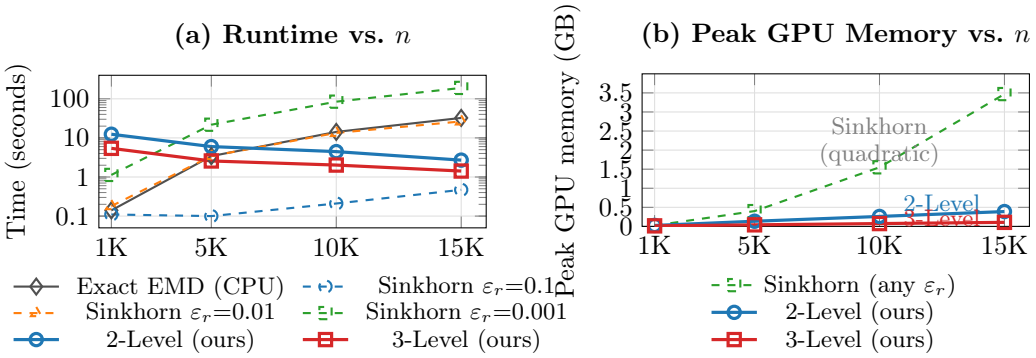


Figure 5: Runtime and peak GPU memory vs. n, synthetic 2D uniform data (seed 42, RTX 2060 SUPER). At n=15,000: our 3-Level solver takes 1.42 s and 0.11 GB vs. Sinkhorn (ε_r=0.001) at 190 s and 3.47 GB. Sinkhorn memory grows quadratically and would exceed 8 GB at $n\approx$45,000; our solvers use $O(n^{3/2})$ and $O(n^{4/3})$ memory and run far beyond this range. Our solvers show decreasing runtime with n because larger point sets produce denser admissible-edge graphs that converge in fewer phases.

in addition to $O(n^2)$ preprocessing. The compressed representation, landmark heaps, and level-2 data structures together use $O(n^{3/2})$ expected space. For fixed ε, the amortized insertion time is $\widetilde{O}(n)$.

D Supplemental Material for Experiment Section

D.1 Comparison with GPU Sinkhorn

We compare against log-domain Sinkhorn (9) from POT (12) on synthetic 2D data. Sinkhorn is swept over $\varepsilon_{\text{reg}} \in \{0.1, 0.01, 0.001\}$; our solvers use ε=0.01. Two caveats: ε_{reg} controls entropic smoothing on a *soft* plan, whereas ε bounds the additive per-pair error of a *hard* assignment; and the cost Sinkhorn reports is a biased overestimate of true OT. Peak GPU memory was measured with `torch.cuda.reset_peak_memory_stats()` and a full cache flush between calls.

At n=15,000, the 3-Level solver runs in 1.42 s and uses 0.11 GB — 134× faster and 32× more memory-efficient than Sinkhorn at ε_r=0.001 (190 s, 3.47 GB). The 2-Level solver takes 2.70 s and 0.39 GB. Because neither solver materialises the $n\times n$ cost matrix, they scale to n=650,000; Sinkhorn runs out of memory at $n\approx$15,000 on this device. Sinkhorn at ε_r=0.1 is fast (under 0.5 s) but the transport cost exceeds 23× the exact value at n=15,000; at ε_r=0.01 the gap exceeds 200%. Figure 5 shows the full runtime and memory comparison; Table 1 summarises the accuracy-time tradeoff.

We compare the memory usage of our compressed representation against log-domain Sinkhorn from POT (9; 12) on synthetic 2D data. Sinkhorn requires explicit construction of the full $n \times n$ cost matrix, leading to quadratic GPU memory growth, while our method stores only the compressed representation. At $n = 15{,}000$, the 3-Level solver uses only 0.11 GB of GPU memory, compared with 3.47 GB for Sinkhorn, making it roughly 32× more memory efficient. The 2-Level solver uses 0.39 GB. Because our approach avoids materializing the full cost matrix, it scales to $n = 650{,}000$ on an 8 GB GPU, whereas Sinkhorn runs out of memory at roughly $n \approx 15{,}000$ on the same device. Figure 5 summarizes the memory comparison.

D.1.1 Comparison with Hierarchical Refinement

We also compare our 2-Level solver (ε=0.01) against Hierarchical Refinement (HiRef) (13), a recent scalable OT method that recursively solves low-rank subproblems to recover a bijective matching. We use the same MNIST equal and biased sampling regimes described earlier. For each problem size, we report the average transport cost returned by the solver

Table 1: Accuracy-time tradeoff on synthetic 2D uniform data, reporting time (s) and absolute gap from exact EMD for each method and problem size. Our solvers produce hard assignments; Sinkhorn costs are soft-plan expected values biased upward relative to true OT. At n=10,000 and n=15,000, both of our solvers are faster than Sinkhorn at ε_r=0.01 and achieve a smaller absolute cost gap.

	$n = 1{,}000$		$n = 5{,}000$		$n = 10{,}000$		$n = 15{,}000$	
Method	time	gap	time	gap	time	gap	time	gap
Sinkhorn ε_r=0.1	0.11	0.1845	0.10	0.2059	0.21	0.2112	0.47	0.2138
Sinkhorn ε_r=0.01	0.18	0.0110	3.45	0.0160	13.42	0.0184	26.40	0.0200
Sinkhorn ε_r=0.001	1.13	3.3×10^{-4}	21.75	1.1×10^{-4}	84.55	8.0×10^{-5}	190.08	4.9×10^{-4}
2-Level (ours)	12.42	0.0070	6.00	0.0071	4.46	0.0072	2.70	0.0079
3-Level (ours)	5.41	0.0238	2.57	0.0130	2.02	0.0117	1.42	0.0121

and the wall-clock runtime on the same RTX 2060 SUPER used throughout the paper. Since both methods output hard matchings, the reported costs are directly comparable.

Figure 4 shows the equal-sampling comparison. The 2-Level solver attains a lower average cost at every tested size while remaining substantially faster. At n=31,569, the 2-Level solver achieves an average cost of 0.051733 compared to 0.058258 for HiRef, and runs in 3.89 s instead of 362.32 s.

Figure 4 shows the biased comparison. In this more difficult setting, HiRef attains a slightly lower average cost across the full range of n, but the gap is small, between 0.3% and 3.3%. The 2-Level solver remains much faster throughout. At n=31,569, HiRef reaches an average cost of 0.081381 versus 0.082596 for the 2-Level solver, while the 2-Level solver runs in 3.36 s compared to 362.64 s for HiRef.

Overall, the comparison shows a clear tradeoff. On similar distributions, the 2-Level solver is both faster and more accurate. On dissimilar distributions, HiRef recovers a modest cost advantage, but at a runtime cost of roughly one to two orders of magnitude.

D.1.2 Accuracy vs. Landmark Density

We vary the landmark multiplier $c \in \{0.5, 1, 2, 4, 8\}$ at fixed n=3,000 on MNIST with equal sampling. Each red point is sampled as a landmark with probability $c/\sqrt{n}$. This changes two parts of the representation in opposite directions: increasing c stores more landmark-to-red distances, while decreasing the number of local blue-red pairs.

Figure 6 shows this tradeoff. In this sweep, smaller c gives lower OT distortion: the 2-Level mean ratio is 1.0056 at c=0.5 and increases to 1.0677 at c=8. The 3-Level proxy follows the same trend, increasing from 1.0216 to 1.1156. On the other hand, the total 2-Level representation size is minimized at c=1, where it stores 315,564 entries and achieves mean ratio 1.0115. We therefore use c=1 as the default setting, since it gives a compact representation while keeping the distortion close to 1.

E k-Level

k-level symmetric construction. Let P be a set of n points. Define a hierarchy

$$P = P_0 \supseteq P_1 \supseteq \cdots \supseteq P_{k-1},$$

where each point in P_{i-1} is sampled independently with probability $n^{-1/k}$ to form P_i.

For each $u \in P$ and each level i, let $p_u^i \in P_i$ be a closest point to u, and define

$$w_u^i = d(u, p_u^i).$$

For each level $i = 0, \ldots, k-2$, define

$$N_u^i = \{v \in P_i : d(u, v) < d(u, p_u^{i+1})\}.$$

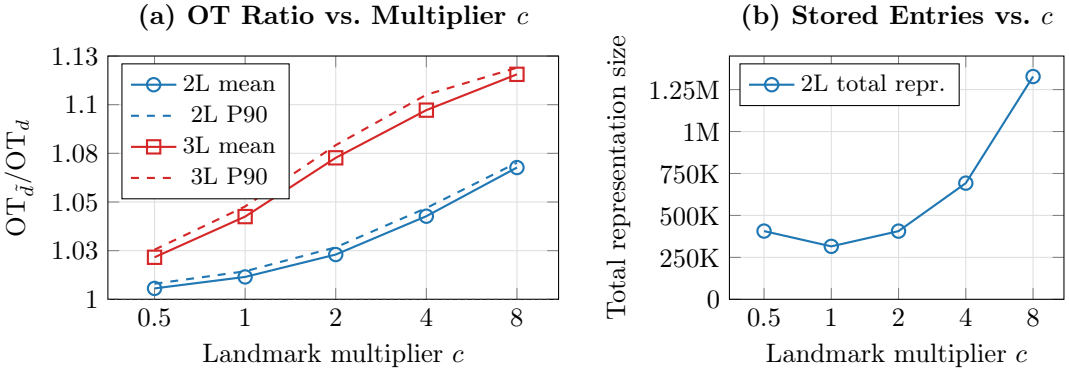


Figure 6: Accuracy and storage vs. landmark multiplier c (MNIST, n=3,000, 10 trials). Solid lines show mean OT ratio; dashed lines show the 90th-percentile ratio. In this sweep, smaller c gives lower OT distortion, while the total representation size is minimized near c=1. The default c=1 therefore provides a balanced operating point, with low distortion and the smallest measured representation size.

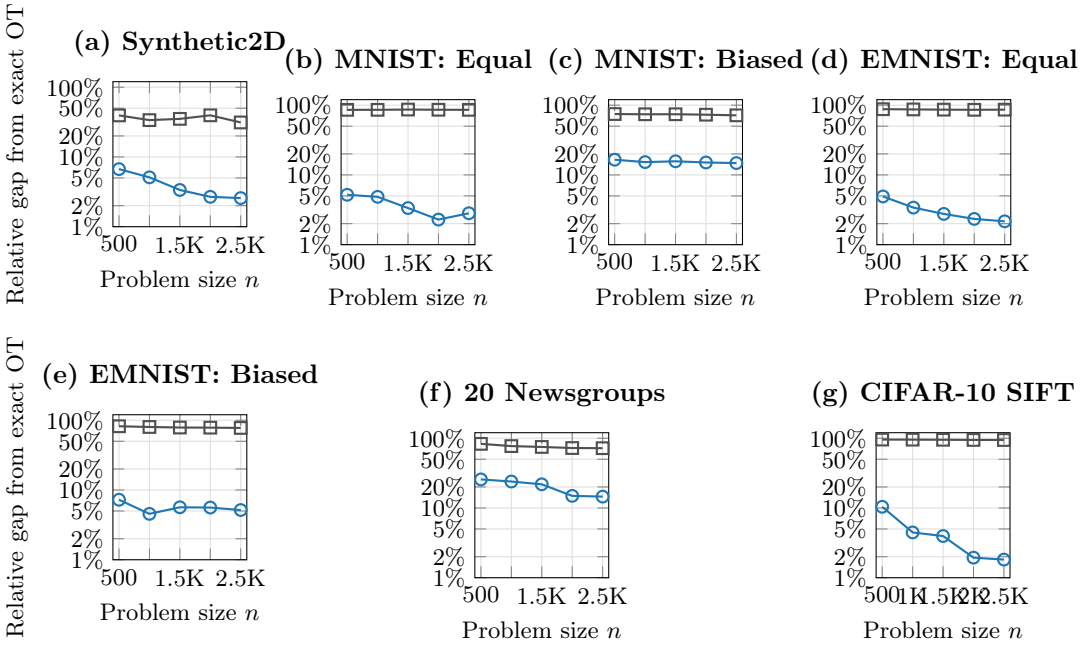


Figure 7: Relative gap from exact OT for the cluster proxy of Agarwal et al. (1) and the 2-Level proxy, $n \in \{500, 1000, 1500, 2000, 2500\}$. Lower is better. The 2-Level proxy is consistently and substantially closer to exact OT across every dataset and every n.

At the top level, define $N_u^{k-1} = P_{k-1}$, and let

$$B(u) = \bigcup_{i=0}^{k-1} N_u^i.$$

Symmetric structured distance. For any two points $u, v \in P$, define

$$d'(u, v) = \min\{d(u, p) + d(v, p)\},$$

where the minimum is taken over all pivots p that satisfy either:

- $p = p_u^i$ for some i and $p \in B(v)$, or

- $p = p_v^i$ for some i and $p \in B(u)$.

Equivalently, $d'(u, v)$ is obtained by routing through the cheapest pivot that is a representative of one endpoint and lies in the bunch of the other endpoint.

Guarantee. For all $u, v \in P$,

$$d(u, v) \le d'(u, v) \le (2k - 1)\, d(u, v).$$

Rank-sensitive interpretation. For each $u \in P$, the construction preserves the nearest $n^{1/k}$ neighbors exactly in expectation. More generally, distances to the nearest $n^{j/k}$ neighbors are preserved within a factor of $2j - 1$ for every $j = 1, \ldots, k$. In particular, the worst-case factor $2k - 1$ is incurred only for the farthest pairs.

Lemma 2. $\mathbb{E}[|B(u)|] = kn^{1/k}$

Proof. Recall that $B(u) := \bigcup_{i=0}^{k-1} N_u^i$. Furthermore, note that N_u^i are disjoint, so we have $|B(u)| = \sum_{i=0}^{k-1} |N_u^i|$. Fix i. Let $x_0, x_1, \ldots, x_{n_i}$ be the points of P_i ordered by non-decreasing distance to u. Each $x \in P_{i+1}$ independently with probability $n^{-1/k}$. Let X be equal to the first index such that $x_X \in P_{i+1}$. Then X is a geometric random variable, so $\mathbb{E}[X] = n^{1/k}$. Note that $x_j \in N_u^i$ only if $j < X$. So $\mathbb{E}[|N_u^i|] = n^{1/k}$.

Therefore, $\mathbb{E}[|B(u)|] = kn^{1/k}$. □

Lemma 3. $\boldsymbol{d}(u, v) \le \boldsymbol{d}'(u, v) \le (2k - 1)\boldsymbol{d}(u, v)$

More generally, if $p_u^j \in B(v)$ or $p_v^j \in B(u)$, then $\boldsymbol{d}(u, v) \le \boldsymbol{d}'(u, v) \le (2j - 1)\boldsymbol{d}(u, v)$.

Proof. $\mathbf{d}'(u, v) = \min\{\mathbf{d}(u, p) + \mathbf{d}(v, p)\}$, where $p = p_u^i \in B(v)$ or $p = p_v^i \in B(u)$.

Note that there it is always such a p because is always the case that $p_u^{k-1} \in B(v)$ and $p_v^{k-1} \in B(u)$.

Let i be the minimum index such that $p_u^i \in B(v)$ or $p_v^i \in B(u)$. Without loss of generality assume that $p_v^i \in B(u)$. Then we have $\mathbf{d}'(u, v) \le \mathbf{d}(u, p_v^i) + \mathbf{d}(v, p_v^i)$. By the triangle inequality,

$$\mathbf{d}'(u, v) \le \mathbf{d}(u, v) + 2\mathbf{d}(v, p_v^i). \tag{4}$$

Note that $\mathbf{d}(u, p_u^0) = 0$ and $\mathbf{d}(v, p_v^0) = 0$. Let $j < i$. We can bound $\mathbf{d}(v, p_v^j)$. Then by assumption, $p_u^j \notin B(v)$ and $p_v^j \notin B(u)$. This implies that $\mathbf{d}(v, p_v^{j+1}) < \mathbf{d}(v, p_u^j)$ and $\mathbf{d}(u, p_u^{j+1}) < \mathbf{d}(u, p_v^j)$. Using the triangle inequality, we have $\mathbf{d}(v, p_v^{i+1}) < \mathbf{d}(u, v) + \mathbf{d}(v, p_v^j)$ and $\mathbf{d}(u, p_u^{j+1}) < \mathbf{d}(u, v) + \mathbf{d}(u, p_u^j)$. It follows that $\mathbf{d}(v, p_v^i) \le i\mathbf{d}(u, v)$

Combining this with Equation 4 gives us $\mathbf{d}'(u, v) \le (2i + 1)\mathbf{d}(u, v)$.

In particular, $i \le k - 1$, so $\mathbf{d}'(u, v) \le (2k - 1)\mathbf{d}(u, v)$ □