Earth Mover's Distance based Similarity Search at Scale

Yu Tang[†], Leong Hou U[‡], Yilun Cai[†], Nikos Mamoulis[†], Reynold Cheng[†]

[†]The University of Hong Kong [‡]University of Macau

[†]{ytang, ylcai, nikos, ckcheng}@cs.hku.hk [‡]ryanlhu@umac.mo

ABSTRACT

Earth Mover's Distance (EMD), as a similarity measure, has received a lot of attention in the fields of multimedia and probabilistic databases, computer vision, image retrieval, machine learning, etc. EMD on multidimensional histograms provides better distinguishability between the objects approximated by the histograms (e.g., images), compared to classic measures like Euclidean distance. Despite its usefulness, EMD has a high computational cost; therefore, a number of effective filtering methods have been proposed, to reduce the pairs of histograms for which the exact EMD has to be computed, during similarity search. Still, EMD calculations in the refinement step remain the bottleneck of the whole similarity search process. In this paper, we focus on optimizing the refinement phase of EMD-based similarity search by (i) adapting an efficient min-cost flow algorithm (SIA) for EMD computation, (ii) proposing a dynamic distance bound, which can be used to terminate an EMD refinement early, and (iii) proposing a dynamic refinement order for the candidates which, paired with a concurrent EMD refinement strategy, reduces the amount of needless computations. Our proposed techniques are orthogonal to and can be easily integrated with the state-of-the-art filtering techniques, reducing the cost of EMD-based similarity queries by orders of magnitude.

1. INTRODUCTION

Given two histograms (e.g., probability distributions), their *Earth Mover's Distance* (EMD) is defined as the minimum amount of work to transform one histogram into the other. EMD is robust to outliers and small shifts of values among histogram bins [20], improving the quality of similarity search in different domain areas, such as computer vision [19, 21], machine learning [6, 9], information retrieval [23, 24], probabilistic [25, 32] and multimedia databases [5, 30]. Typically, the EMD between two histograms is modeled and solved as a linear optimization problem, the *min-cost flow problem*, which requires super-cubic time. The high computational cost of EMD restricts its applicability to datasets of low-scale. For example, in computer vision applications, the quality of results is typically compromised by the use of low-granularity histograms, to render EMD-based similarity retrieval feasible [22, 30].

Proceedings of the VLDB Endowment, Vol. 7, No. 4

EMD-based similarity search has been extensively studied in [5, 25, 30, 32]. Given a query histogram $\mathbf{q}$ and a database of histogram objects $\mathcal{D}$, the objective is to find the k nearest neighbors of $\mathbf{q}$ in $\mathcal{D}$. All these works adopt the *filter-and-refinement* framework; to evaluate a query, unpromising objects (or object groups) are filtered out, by utilizing various effective EMD lower bound estimations, based on centroids and projections [7], dimensionality reduction [30], primal-dual space [32], normal distributions [25], etc. Actual EMD calculations[1] are applied only between $\mathbf{q}$ and all objects that survive the filter step. Thus, the primary focus of previous research has been in tightening the lower bounds such that more objects can be pruned at the filter step. For instance, [25] demonstrate that the projection-based lower bound can be up to 90% of the actual EMD. However, the effectiveness of a lower bound largely depends on various factors, such as the dimensionality and granularity of histograms, the data distribution, and the parameters of the similarity query (e.g., k). In particular, for large-scale datasets (e.g., 1M cardinality and/or 1K histogram dimensionality), the current state-of-the-art solution [25] is not feasible, due to the extreme cost of the refinement step. For instance, based on the experiments in [25], it may take 10 minutes[2] to complete one k-NN query on a dataset with 1M objects even when 99% of objects are filtered out.

In view of this, we focus on optimizing the *refinement phase* of EMD-based similarity search. Calculating the EMD between two object histograms is equivalent to finding the *minimum-cost flow* (MCF) in a bipartite network, where each vertex indicates a histogram bin and edges connect bins from different object histograms. Techniques from operations research [1], such as network simplex, primal-dual, successive shortest path, and cost-scaling can be used to solve MCF. However, these solutions do not scale well with the number of histogram bins since their computations rely on a complete bipartite network. For example, consider two histograms having 1K bins each, and the corresponding flow graph (bipartite network) with 1M (1K $\times$ 1K) edges in total. Constructing and using this graph for solving MCF requires high computational resources. To alleviate this problem, we adapt a *simplified graph incremental algorithm* (SIA), originally proposed for assignment-joins in spatial databases [29], which incrementally constructs the flow graph during the flow calculations based on the edge weights. Our adaptation significantly reduces the EMD computation time.

Min-cost flow algorithms, such as SIA, only aim at efficiently evaluating a single EMD computation but they do not exploit the execution plan of EMD-based similarity queries. In other words, by integrating SIA into the filter-and-refinement framework as a black-box module, the number of EMD calculations is not affected,

[1]By *EMD calculation* we refer to the entire run of an algorithmic process that computes the EMD between two histograms.
[2]A linear estimation derived from the IRMA experiment in [25].

and every calculation is still conducted at its *entirety*. In our study, we observe that it is possible to incrementally derive and tighten a *running lower bound* for the EMD during the SIA calculation. Based on this, we introduce a *progressive bounding* (PB) technique, which can terminate the SIA calculations early for histograms that are no longer promising to the similarity query. In addition, we propose a *dynamic refinement ordering* (DRO) technique, which concurrently handles and dynamically re-orders multiple EMD calculations. These two techniques greatly reduce the computations at the refinement phase of EMD-based similarity search, boosting the search performance.

PB and DRO can be seamlessly integrated with any existing (and future) filtering techniques. We show by experimentation that our techniques can compute EMD-based similarity queries one to two orders of magnitude faster, compared to the current state-of-the-art [25]. To the best of our knowledge, ours is the first study on this subject that considers datasets of million-scale on the object cardinality and thousand-scale on the histogram dimensionality.

The rest of the paper is organized as follows. Section 2 formally defines EMD, presents a min-cost flow algorithm for its computation, and discusses the standard filter-and-refinement framework used for EMD-based similarity queries. Section 3 describes SIA, an optimized implementation of the successive shortest path MCF algorithm. Section 4 presents our progressive bounding and dynamic refinement ordering techniques. Section 5 includes an extensive experimental evaluation which demonstrates the effectiveness of our techniques. Related work is presented in Section 6. Finally, Section 7 concludes the paper with a discussion about future work.

2. PRELIMINARIES

The *Earth Mover's Distance* (EMD), first introduced by the computer vision community in [23,24], is a distance function that measures the dissimilarity of two histograms (e.g., probability or feature distributions). Given two histograms $\mathbf{q} = (q_1, \ldots, q_n)$ and $\mathbf{p} = (p_1, \ldots, p_n)$, each having n bins, a *flow matrix* $\mathbf{F}$, where $f_{i,j}$ indicates flow (i.e., earth) to move from q_i to p_j, and a *cost matrix* $\mathbf{C}$, where $c_{i,j}$ models cost of moving flow from the i-th bin to the j-th bin, we can define the total cost of moving unit flow according to $\mathbf{F}$ and $\mathbf{C}$ between $\mathbf{q}$ and $\mathbf{p}$ as

$$d(\mathbf{q}, \mathbf{p}) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{i,j} c_{i,j} \quad (1)$$

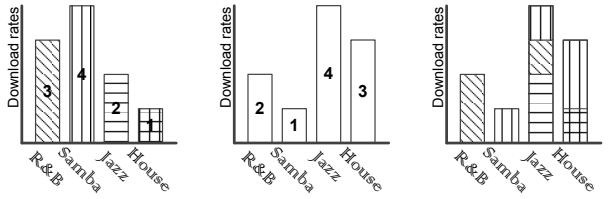
The cost matrix (a.k.a., ground distance) $\mathbf{C}$ can be designed by domain experts and/or derived from a mathematical formula [25, 32]. Intuitively, $c_{i,i} = 0$ and the larger the distance between i and j in the bin space, the larger $c_{i,j}$ is.[3] Assuming that $\mathbf{q}$ and $\mathbf{p}$ are *normalized* such that $\sum_{i=1}^{n} q_i = \sum_{i=1}^{n} p_i$, the EMD between $\mathbf{q}$ and $\mathbf{p}$ is formally defined as follows:

$$\begin{aligned} emd(\mathbf{q}, \mathbf{p}) = & \min_{\mathbf{F}} d(\mathbf{q}, \mathbf{p}), \\ \text{such that} \quad & \forall i, j \in [1, n] : f_{i,j} \geq 0, \\ & \forall i \in [1, n] : \sum_{j=1}^{n} f_{i,j} = q_i, \\ \text{and} \quad & \forall j \in [1, n] : \sum_{i=1}^{n} f_{i,j} = p_j \end{aligned} \quad (2)$$

[3]The motivating example of [32] partitions a 2-dimensional feature space (humidity and temperature) into 4×4 cells based on the range of domain values. The cost $c_{i,j}$ between bins i and j is represented by their Euclidean distance of the corresponding cells.

$emd(\mathbf{q}, \mathbf{p})$ is the minimum cost needed to transform $\mathbf{q}$ to $\mathbf{p}$; to do so, we distribute the flow (i.e., earth) from each bin q_i to a set of initially empty bins for $\mathbf{p}$, such that the resulting histogram will be equal to $\mathbf{p}$. As moving earth $f_{i,j}$ from q_i to the j-th bin of $\mathbf{p}$ bears a cost $f_{i,j}c_{i,j}$, the objective is to find the flow distribution that results in the minimum total cost. Note that $emd(\mathbf{q}, \mathbf{p})$ is equal to $emd(\mathbf{p}, \mathbf{q})$ when the cost matrix $\mathbf{C}$ is symmetric.

We demonstrate the calculation and applicability of EMD via a real example from web data analysis. Figure 1(a) and 1(b) illustrate the download rates of four music genres by two customers, $\mathbf{q}$ and $\mathbf{p}$, in an online store. The rates are normalized such that all values of each histogram sum to 10. To calculate the distance $emd(\mathbf{q}, \mathbf{p})$ between the two customers, we should identify the minimum work to transform genre distribution $\mathbf{q}$ to distribution $\mathbf{p}$. Assume that the cost matrix $\mathbf{C}$ of the music genres is as shown in Table 1, where indices 1, 2, 3, 4 denote the four music genres (i.e., *R&B*, *Samba*, *Jazz*, and *House*, respectively). Figure 1(c) illustrates the best transformation of $\mathbf{q}$ to $\mathbf{p}$ in terms of the total cost among all feasible transformations. For instance, there are 3 units in $\mathbf{q}$'s *R&B* genre. In the transformation, $f_{1,1} = 2$ units are moved to $\mathbf{p}$'s *R&B* (with cost $2 \cdot c_{1,1} = 0$) and $f_{1,3} = 1$ unit is moved to $\mathbf{p}$'s *Jazz* (with cost $1 \cdot c_{1,3} = 0.1$). Thus, based on the best transformation, $emd(\mathbf{q}, \mathbf{p})$ is $0.1 + 2.4 + 0 + 0 = 2.5$, providing a quantitative measure for the dissimilarity between these two customers. This example demonstrates an application of EMD to viral marketing analysis, which enables enterprises to derive similarities between customers in order to promote their products.

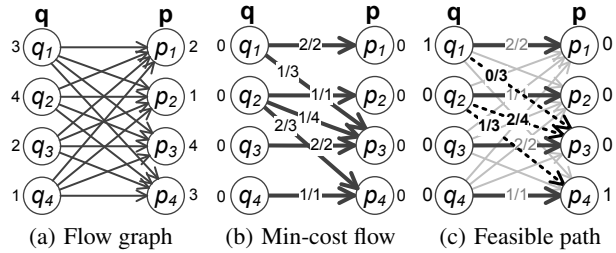


(a) $\mathbf{q}$'s distribution (b) $\mathbf{p}$'s distribution (c) $emd(\mathbf{q}, \mathbf{p})$

Figure 1: A concrete example of online music library analysis

Table 1: Cost matrix C of 4 music genres

	p_1	p_2	p_3	p_4
q_1	0	0.9	0.1	0.7
q_2	0.9	0	0.6	0.9
q_3	0.1	0.6	0	0.3
q_4	0.7	0.9	0.3	0



(a) Flow graph (b) Min-cost flow (c) Feasible path

Figure 2: The flow network of the music example

2.1 Computing the EMD

EMD can be computed using linear programming [12] and network flow algorithms [1]. We now explain how EMD computation can be modeled and solved as a network flow problem. We first construct a bipartite flow network (see Figure 2(a) for the example of Figure 1), where the vertices are derived from the histogram bins (e.g., music genres) and the edges connect the bins between the two histograms. Each edge carries a cost according to the corresponding cell of the cost matrix. The *flow capacity* of each vertex corresponds to the value of the corresponding histogram bin. For instance, the flow capacity of vertex q_1 (i.e., *R&B* of $\mathbf{q}$) in Figure 2(a) is set to 3 according to Figure 1(a). Finding the *minimum-cost flow* in this bipartite graph is equivalent to finding the EMD from $\mathbf{q}$ to $\mathbf{p}$. Each vertex of $\mathbf{q}$ should send total flow equal to its capacity and each vertex of $\mathbf{p}$ should receive total flow equal to its capacity. The minimum-cost flow is shown in Figure 2(b). On each edge $e(q_i, p_j)$, the label $f_{i,j}/cap_{i,j}$ shows the flow $f_{i,j}$ sent from the origin vertex q_i and the *capacity* $cap_{i,j}$ of that edge (i.e., the maximum flow which could possibly be sent from q_i to p_j). The edge capacity $cap_{i,j}$ is the minimum capacity of the two end-vertices; e.g., the capacity of $e(q_2, p_4)$ is 3 $(= \min\{cap_{q_2}, cap_{p_4}\} = \min\{4, 3\})$.

The *successive shortest path* (SSP) algorithm [1] is the most representative algorithm in the category of the network flow based solutions. SSP iteratively computes and *augments* paths that (i) start from a vertex q_i which has remaining flow capacity, (ii) terminate to a vertex p_i which also has remaining flow capacity, and (iii) nodes from $\mathbf{q}$ and $\mathbf{p}$ are alternated in these paths. A valid path should include *feasible* edges only. Given a flow graph, an edge is *feasible* if there is remaining flow capacity on the edge. When augmenting a flow $f_{i,j}$ on an edge $e(q_i, p_j)$, we subtract $f_{i,j}$ from the capacity of $e(q_i, p_j)$ and add $f_{i,j}$ to the flow capacity of $e(p_j, q_i)$. In our running example, initially, no flow has been augmented on any edge (i.e., the $f_{i,j}$ labels of all edges are set to 0); thus, edge $e(q_2, p_4)$ (illustrated in Figure 3(a)) is feasible since the remaining flow capacity from q_2 to p_4 is 3 (illustrated by the number on the dashed line). If we augment 1 unit of flow on $e(q_2, p_4)$, we subtract 1 from the capacity of $e(q_2, p_4)$ and add 1 to the capacity of $e(p_4, q_2)$ (as shown in Figure 3(b)). Note that $e(p_4, q_2)$ is not a physical edge in G, as there are only directed edges from $\mathbf{q}$ to $\mathbf{p}$ but not vice-versa. However, during path computation, SSP traverses also reverse edges provided that they are feasible. A formal definition of feasible edges is given below. The capacity of a non-physical edge (i.e., reverse edge) $e(p_j, q_i)$ always equals to the flow $f_{i,j}$ currently on edge $e(q_i, p_j)$.

DEFINITION 1 (FEASIBLE EDGE). *Given a flow graph, a physical edge $e(q_i, p_j)$ is feasible if $f_{i,j} < cap_{i,j}$; a non-physical (reverse) edge $e(p_j, q_i)$ is feasible if $f_{i,j} > 0$.*

(a) Before augmentation (b) After augmentation

Figure 3: Augmenting 1 flow on edge $e(q_s, p_h)$

Besides, the *cost* $c_f(u, v)$ of an edge $e(u, v)$ is determined by its physical existence in the flow graph and the cost matrix $\mathbf{C}$:

DEFINITION 2 (COST OF FEASIBLE EDGE). *The cost of a physical edge $e(q_i, p_j)$ is $c_f(q_i, p_j) = c_{i,j}$, while the cost of a non-physical (reverse) edge $e(p_j, q_i)$ is $c_f(p_j, q_i) = -c_{i,j}$.*

For instance, the reverse edge $e(p_4, q_2)$ in Figure 3(b) is feasible. Its cost is $c_f(p_4, q_2) = -c_{2,4} = -0.9$ since $e(p_4, q_2)$ is not a physical edge. To calculate the min-cost flow (i.e., EMD), SSP iteratively searches for the *feasible* path having the *minimum cost*. As discussed above, a feasible path starts from a vertex in $\mathbf{q}$, which still has positive flow capacity, ends at a vertex in $\mathbf{p}$ with positive flow capacity and includes only feasible edges. For instance, there is a feasible path highlighted by three dashed lines in Figure 2(c); the path starts at q_1, passes p_3 and q_2, and finally reaches p_4. The cost of this path is $c_f(q_1, p_3) + c_f(p_3, q_2) + c_f(q_2, p_4) = 0.1 + (-0.6) + 0.9 = 0.4$. SSP selects the feasible path with the *lowest cost* and *augments* the maximum possible flow along the path. The augmented flow is determined by the minimum of the following quantities: (i) the remaining flow capacity at the source node, (ii) the remaining flow capacity at the destination, (iii) the minimum remaining capacity of all edges on the path. For example, Figure 2(b) shows the result of augmenting the path shown by the three dashed lines in Figure 2(c). The augmentation adds 1 flow unit to all physical edges on the path (i.e., $e(q_1, p_3)$ and $e(q_2, p_4)$), subtracts 1 flow unit from the edges, for which the reverse edge is on the path (i.e., $e(q_2, p_3)$), and updates the capacities of path edges, q_1, and p_4.

Computing the EMD, using SSP requires $O(F|E|log|V|)$ time, where F is the total number of flows we need to augment and $O(|E|log|V|)$ is the cost of a shortest path search on a bipartite graph with $|V|$ vertices and $|E|$ edges. After each path augmentation, the *changes in the graph* render the subsequent shortest path search *independent* from the previous one, therefore, a large number of shortest path computations should be applied. As we shall see in Section 3, we can greatly reduce the cost of SSP by a method which builds and searches the flow graph incrementally.

2.2 Filter-and-Refinement Framework

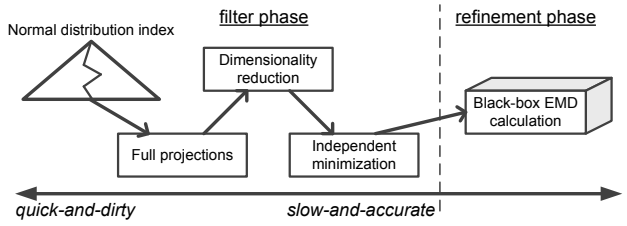


Figure 4: Filter-and-refinement framework

Given a collection $\mathcal{D}$ of histograms and a *query* histogram $\mathbf{q}$, a k-nearest neighbor (k-NN) query, finds a subset $\mathcal{S}$ of $\mathcal{D}$ containing k histograms, such that $\forall \mathbf{p} \in \mathcal{S}, \forall \mathbf{p}' \in \mathcal{D} \setminus \mathcal{S}, emd(\mathbf{q}, \mathbf{p}) \leq emd(\mathbf{q}, \mathbf{p}')$. The k-NN query is the most popular similarity search type, as the number of results is controlled by k and there is no requirement for setting a similarity threshold prior to search. In previous studies [5, 7, 25, 30], k-NN queries are evaluated based on a filter-and-refinement framework. The EMD $emd(\mathbf{q}, \mathbf{p})$ between the query histogram $\mathbf{q}$ and every histogram $\mathbf{p} \in \mathcal{D}$ is estimated with the help of lower bound filtering techniques, such as the *normal distribution index* [25], the *full projection lower bound* [7], the *reduced dimension lower bound* [30], and *independent minimization* [5]. In general, these filters are applied in an order starting from *quick-and-dirty* ones to *slow-and-accurate* ones, as shown in Figure 4. For histograms $\mathbf{p}$ that cannot be pruned by the filters, the actual $emd(\mathbf{q}, \mathbf{p})$ is calculated by a black-box computation module, such as SSP or transportation simplex [12].

Algorithm 1 FILTER-AND-REFINEMENT FOR k-NN

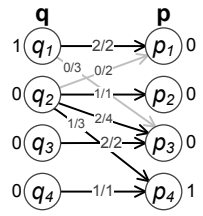
H: heap, θ: pruning threshold
Algorithm k-NN(Query $\mathbf{q}$, Index I, Filters Δ)
1: $\theta := \infty$; $H := \emptyset$
2: **while** $I.getnext(\mathbf{q}, \theta, \langle \mathbf{p}, lb_{\mathbf{p}} \rangle)$ **do**
3: **for** $\delta_i \in \Delta$ **do** ▷ Filter phase
4: $lb_{\mathbf{p}} := \max\{lb_{\mathbf{p}}, \delta_i(\mathbf{q}, \mathbf{p})\}$
5: **if** $lb_{\mathbf{p}} \geq \theta$ **then** break loop
6: **if** $lb_{\mathbf{p}} < \theta$ **then** ▷ Refinement phase
7: **if** $emd(\mathbf{q}, \mathbf{p}) < \theta$ **then**
8: update H to include $\langle \mathbf{p}, emd(\mathbf{q}, \mathbf{p}) \rangle$
9: $\theta :=$ k-th EMD value in H
10: **return** H

Algorithm 1 is a pseudocode of the filter-and-refinement framework used in previous work. Since histograms correspond to objects (e.g., images), we will use the terms objects and histograms interchangeably. First, the framework accesses objects from an index I, such as the *normal distribution index* [25] or the *TBI index* [32]. These indexes provide a $getnext$ function which returns at each call an unseen object $\mathbf{p}$ having lower bound $lb_{\mathbf{p}}$ of $emd(\mathbf{q}, \mathbf{p})$ smaller than a given threshold θ (line 2). In k-NN search, θ is the k-th largest EMD computed so far (i.e., the distance of the current k-th NN of $\mathbf{q}$ in $\mathcal{D}$). At each iteration (lines 2–9), the framework accesses a histogram $\mathbf{p} \in \mathcal{D}$ using the $getnext$ function and attempts to tighten its lower bound $lb_{\mathbf{p}}$ by applying a set Δ of progressively more expensive and accurate lower bound estimation techniques (see Figure 4). If any of the computed lower bounds is not smaller than the pruning threshold θ (line 5), then $\mathbf{p}$ is *filtered*, i.e., the exact $emd(\mathbf{q}, \mathbf{p})$ needs not to be computed. Otherwise, $emd(\mathbf{q}, \mathbf{p})$ is essentially computed by a black-box algorithm (line 7). During search, Algorithm 1 maintains a heap H with the k histograms having the lowest EMD so far and the pruning threshold θ (lines 8–9). The k-NN candidates are confirmed as the result, when no more objects are returned by the $getnext$ function (line 2), i.e., all unseen objects do not satisfy the distance threshold θ.

3. SCALING UP SSP

Computing EMD by SSP requires having the complete bipartite graph between $\mathbf{q}$ and $\mathbf{p}$, which is quadratic to the number of histogram bins. In our previous work, we proposed an optimized version of SSP, *simplified graph incremental algorithm* (SIA) [29], to scale up the computation of *spatial matching* problems. We now show how SIA can be adopted to scale up the computation of EMD, for histograms with a large number of bins. Different from SSP, SIA incrementally constructs a *partial* flow graph G' by inserting edges from the complete bipartite graph G to G'. The incremental graph construction significantly reduces the search cost since the size of the partial graph G' is typically much smaller than the complete graph G. We use the running example (cf. Figure 2) to demonstrate the superiority of SIA in Figure 5. Recall that the min-cost feasible path starting at q_1 (cf. Figure 2(c)) is $q_1 \rightarrow p_3 \rightarrow q_2 \rightarrow p_4$ in the complete graph G. Suppose a partial graph G' is constructed based on the seen values in the partial cost matrix (Figure 5(b)), the same min-cost feasible path can be found in G' as well. Thereby, augmenting this path in G' is equivalent to the augmentation in G which returns the same result in Figure 2(b).

The question now turns to *how we can guarantee that the min-cost feasible paths in G and G' are equivalent*. This is done by a distance bound checking where the bound Π is derived from the edges not yet inserted into G'. As an intuition, if the edges in G' are inserted incrementally in ascending order to their costs at every vertex, it is possible that the min-cost path in G' is cheaper than all



(a) Partial graph, G'

	p_1	p_2	p_3	p_4
q_1	0	-	0.1	-
q_2	0.9	0	0.6	0.9
q_3	-	-	0	-
q_4	-	-	-	0

(b) The cost matrix of G'

Figure 5: SIA on a partial flow graph

feasible paths which include edges not in G'. In this case, the path is augmented; otherwise SIA expands G' by inserting more edges until the best feasible path in G' has lower cost than the threshold Π[4]; the expansion of G' can only reduce the cost of the feasible flow and increase Π, as edges are inserted to G' in increasing order of their costs.

Algorithm 2 SIA BASED EMD CALCULATION

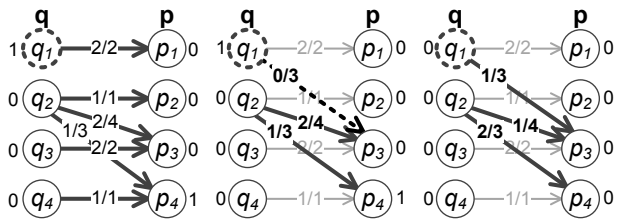
Π: distance bound, G': running subgraph
Algorithm emd_{SIA}(Histograms $\mathbf{q}$, $\mathbf{p}$)
1: $\Pi := 0$; $G' := \emptyset$
2: **while** $\exists$ feasible q_i **do**
3: $sp :=$Dijkstra(q_i, G')
4: **while** $sp.cost > \Pi$ or sp doesn't reach any feasible p_j **do**
5: insert min-cost edge $e(q_l, p_m) \in G - G'$ into G'
6: update distance bound Π
7: $sp :=$Dijkstra(q_i, G')
8: augment sp
9: **return** total augmenting cost

Algorithm 2 is a pseudocode of SIA for EMD calculations. At each iteration, SIA searches the min-cost feasible path sp using Dijkstra's shortest path algorithm [1] in G' from any feasible vertex (line 3), i.e., a vertex of $\mathbf{q}$ with remaining capacity.[5] If the cost of sp does not exceed the distance bound Π (line 4), then it must be a valid min-cost feasible path in the entire graph G. We augment the flow of sp if sp is valid (line 8). Otherwise, G' is essentially expanded by adding more edges from G (line 5) and the distance bound Π is updated accordingly (line 6).

We demonstrate the functionality of SIA by the example of Figure 6. Suppose that G' contains only 6 edges and there are 9 flow capacities sent already. According to the flow capacities, q_1 is the only feasible node in $\mathbf{q}$ but there is no feasible path currently from q_1 to any node of $\mathbf{p}$ (see Figure 6(a)). Subsequently, we insert a new edge, $e(q_1, p_3)$, into G' and now there is a shortest path, $sp = \langle e(q_1, p_3), e(p_3, q_2), e(q_2, p_4) \rangle$. The cost of sp is 0.4, which is smaller than the distance bound Π, thus we return sp as the result of the current search and augment 1 unit of flow from q_1 and q_2 to p_3 and p_4, while 1 unit of flow from q_2 to p_3 is canceled.

[4]For clarity, $\Pi = c_{max}(E - E') - \tau_{max}$, where $c_{max}(\cdot)$ returns the maximum cost in a set of edges, E' denotes the edges in G', and τ_{max} indicates the largest potential value of the vertices. As a note, Π can be further tightened if considering only a subset of E' where the edges are from the vertices being *visited* by the current Dijkstra search. The correctness proof and optimization details are given in [29].

[5]Since G (and G') may contain edges of negative costs (i.e., the reverse of edges that currently carries flow), Dijkstra's algorithm cannot be directly applied. To make its application possible, we need to iteratively maintain a *potential* value at every vertex, which transforms the costs of the feasible edges to non-negative values. The details (see [29]) are omitted for the sake of readability.



(a) G' at loop i (b) G' at loop $i+1$ (c) G' at loop $i+2$

Figure 6: A running example of SIA-EMD

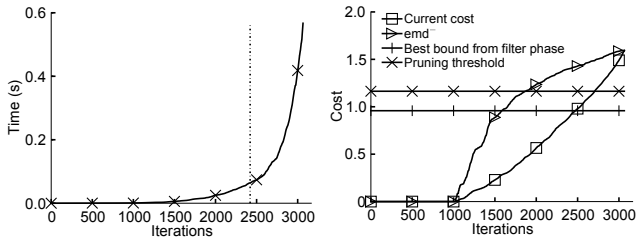
In this work, we use SIA as a module for computing the EMD between two histograms $\mathbf{p}$ and $\mathbf{q}$ in the refinement step of similarity search, due to its efficiency and scalability to the number of histogram bins. Still, SIA only optimizes the performance of an individual EMD calculation, whereas our ultimate objective is to minimize the overall cost of a similarity query, which may involve a large number of EMD calculations. The next section shows how we can optimize the overall cost of queries by exploiting information during the course of a SIA calculation.

4. BOOSTING THE REFINEMENT PHASE

In this section, we propose two novel techniques, *progressive bounding* (PB) and *dynamic refinement ordering* (DRO), which boost the performance of the refinement phase during EMD-based similarity search. PB is inspired by the running time pattern of a single EMD calculation and DRO is inspired by the execution order in the filter-and-refinement framework. These two techniques make the EMD calculations at the refinement phase being handled in multiple stages and progressively instead of as one-off processes (i.e., by black-box modules).

4.1 Analysis of EMD Calculation

According to previous experimental studies (e.g., [25, 32]), more than 95% of the histograms on average can be filtered by the lower bound estimations; however, such high filter effectiveness is not guaranteed. To make things worse, if we process EMD-based similarity queries on large datasets (e.g., 1M objects) having high-granularity histograms (e.g., several hundreds of bins), the refinement phase even at a very high filtering ratio (e.g., 99%) easily becomes the bottleneck, due to the high cost of exact EMD calculations. Although there has been ample work on improving the performance and effectiveness of the filter phase in EMD-based similarity search, to the best of our knowledge, there has been no work focusing on optimizing the *refinement phase*, for which off-the-shelf EMD computation techniques (such as transportation simplex or SSP) are simply applied as black-box modules.



(a) SIA's cumulative runtime (b) Current cost

Figure 7: SIA performance on different iterations

As discussed in Section 3, we adopt SIA for exact EMD calculations in the refinement phase. Although SIA offers great performance improvements over typical EMD solutions (e.g., SSP [1] and transportation simplex [12]), the running time during its execution increases quickly as the algorithm progresses. Figure 7(a) shows the cumulative runtime over SIA iterations during a typical EMD calculation for two histograms with 1024 bins each. This example shows that over 90% of SIA's execution time is used in the last 20% of its iterations. The reason behind this is that as SIA progresses, the partial graph G' and the number of feasible edges grow and shortest path searches become much more expensive. This analysis shows that there is room for greatly improving the refinement phase of EMD-based similarity search, if SIA can be terminated before reaching its late iterations.

4.2 Progressive Bounding

In order to terminate SIA as early as possible for objects that do not make it to the query result set, we propose a technique which *progressively* maintains a *running lower bound* emd^- and tightens emd^- throughout the entire EMD calculation process. SIA, for a specific refinement, can terminate early if the running lower bound becomes not smaller than the current pruning threshold θ (i.e., the k-th lowest EMD found so far).

First, we use a property of SSP (and SIA); at each iteration, the minimum-cost feasible path is nonnegative when the cost matrix $\mathbf{C}$ does not include any negative values [1]. In other words, the accumulated cost by augmenting flows at each iteration is monotonically non-decreasing, and can be used as the running lower bound of EMD. For example, Figure 7(b) illustrates the accumulation of the EMD during SIA, for two 1024-bin histograms $\mathbf{q}$ and $\mathbf{p}$. At each iteration, the *current cost* represents the accumulation of the augmented costs up to that iteration; by comparing it with the current pruning threshold θ (ignore the other series for the moment), we can observe that SIA may terminate as soon as the current cost becomes at least equal to θ (i.e., at around 88.3% of the total iterations). This point corresponds to 23.4% of SIA's total execution time (cf. Figure 7(a)). Note that the current cost remains 0 until the 1024th iteration because the shortest paths up to this iteration correspond to flows between the same bins of $\mathbf{q}$ and $\mathbf{p}$ (i.e., from q_i to p_i for some $i \in [1, n], n = 1024$) with zero cost each.

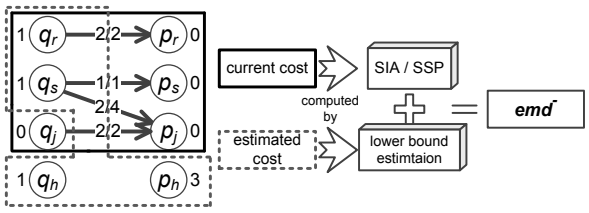


Figure 8: Running lower bound, emd^-

Although the *current cost* could be intuitively used as a lower bound for the final (actual) EMD, it is not sufficiently tight since it does not take the remaining feasible flow into consideration. Thereby, we propose a *running lower bound*, emd^-, which not only considers the so-far accumulated flow cost, but also provides an estimation to the cost of flows yet to be augmented. As Figure 8 illustrates, emd^- consists of the *current cost* (partial EMD already computed) plus a lower bound for the cost of the remaining flow. The effectiveness of emd^- is shown in the running example of Figure 7(b), where emd^- grows much faster compared to the

current cost and leads to an early termination of SIA for EMD calculations. In this example, the running lower bound emd^- reaches the pruning threshold θ after augmenting 61% iterations which take only 2.84% of the total execution time. This result demonstrates the effectiveness of emd^- in boosting EMD-based similarity search.

We now formally define emd^-. First, Lemma 1 shows that the costs of the minimum-cost feasible paths being augmented from a vertex are monotonically non-decreasing, which forms a basis for the definition of the bound.

LEMMA 1 (COST MONOTONICITY OF FEASIBLE PATHS). *Given a flow graph for which there is a sequence of m_i minimum-cost feasible paths $sp_{q_i,1}, \ldots, sp_{q_i,m_i}$ from a node $q_i \in \mathbf{q}$ already been augmented in this order, the cost of these paths from q_i is monotonically non-decreasing, i.e., $c_f(sp_{q_i,j}) \leq c_f(sp_{q_i,k}), \forall 1 \leq j < k \leq m_i$.*

PROOF. To prove the statement, it suffices to show that the cost of the shortest path from any $q_i \in \mathbf{q}$ to any $p_j \in \mathbf{p}$ is monotonically non-decreasing throughout the entire flow augmentation process; i.e., $c_f^{t-1}(q_i \rightsquigarrow p_j) \leq c_f^t(q_i \rightsquigarrow p_j)$, where c_f^t is the cost of the shortest path from q_i to p_j after t augmentations. If this holds, then the cost of the minimum-cost feasible paths from q_i must be monotonically non-decreasing, because at each iteration, SIA picks for q_i the path with the minimum cost, and there is always a feasible path between any pair of nodes with remaining capacity.

We prove this by contradiction. Assume that the monotonicity of the shortest path cost from q_i to a vertex p_j does not hold, i.e., $c_f^{t-1}(q_i \rightsquigarrow p_j) > c_f^t(q_i \rightsquigarrow p_j)$, and assume that no path violates the monotonicity property until the t-th iteration. In addition, among all paths that violate the monotonicity property at the t-th iteration, $c_f^t(q_i \rightsquigarrow p_j)$ has the minimum cost. Suppose that q_k is the preceding vertex of p_j in the shortest path, we have

$$c_f^t(q_i \rightsquigarrow p_j) = c_f^t(q_i \rightsquigarrow q_k) + c_f^t(q_k, p_j) \geq c_f^t(q_i \rightsquigarrow q_k).$$

In addition, based on our assumption, no vertex violates the monotonicity property before the t-th augmentation. Thereby,

$$c_f^{t-1}(q_i \rightsquigarrow q_k) \leq c_f^t(q_i \rightsquigarrow q_k)$$

By combining the equations, we get

$$c_f^{t-1}(q_i \rightsquigarrow p_j) > c_f^t(q_i \rightsquigarrow p_j) \geq c_f^t(q_i \rightsquigarrow q_k) \geq c_f^{t-1}(q_i \rightsquigarrow q_k)$$

If $e(q_k, p_j)$ is feasible in G^{t-1}, then:

$$\begin{aligned} c_f^{t-1}(q_i \rightsquigarrow p_j) &\leq c_f^{t-1}(q_i \rightsquigarrow q_k) + c_f^{t-1}(q_k, p_j) \\ &\leq c_f^t(q_i \rightsquigarrow q_k) + c_f^t(q_k, p_j) = c_f^t(q_i \rightsquigarrow p_j); \end{aligned}$$

otherwise ($e(p_j, q_k)$ is feasible in G^{t-1}):

$$\begin{aligned} c_f^{t-1}(q_i \rightsquigarrow q_k) &\geq c_f^{t-1}(q_i \rightsquigarrow p_j) + c_f^{t-1}(q_k, p_j) \\ &> c_f^t(q_i \rightsquigarrow p_j) + c_f^t(q_k, p_j) \geq c_f^t(q_i \rightsquigarrow q_k). \end{aligned}$$

Both cases contradict our assumptions. □

DEFINITION 3 (RUNNING LOWER BOUND, emd^-). *Consider a flow graph, such that for each node $q_i \in \mathbf{q}$, there is a set of m_i augmented paths $sp_{q_i,1}, \ldots, sp_{q_i,m_i}$. The running lower bound emd^- is defined as*

$$emd^- = \sum_{q_i \in \mathbf{q}} \sum_{j=1}^{m_i} c_f(sp_{q_i,j}) \cdot f(sp_{q_i,j}) + \quad \textbf{(current cost)}$$

$$\sum_{q_i \in \mathbf{q}} c_f(sp_{q_i,m_i}) \cdot (cap_{q_i} - f_{q_i}), \quad \textbf{(estimated cost)}$$

where cap_v indicates the total flow capacity of node v, and f_v indicates the flow units already augmented from v so far.

LEMMA 2 (CORRECTNESS OF emd^-). *The running lower bound emd^- is monotonically non-decreasing and always not greater than $emd(\mathbf{q}, \mathbf{p})$ throughout the EMD calculation.*

PROOF. Trivial, due to the monotonicity of the shortest path costs from any vertex q_i (Lemma 1) and due to the fact that all remaining flow $cap_{q_i} - f_{q_i}$ at q_i should be augmented in paths originating at q_i. □

Table 2: Four augmented paths and their costs

iteration	feasible path	$c_f(sp_i)$	$f(sp_i)$	current cost
1	$sp_1 = \langle e(q_1, p_1) \rangle$	0.0	2	0.0
2	$sp_2 = \langle e(q_3, p_3) \rangle$	0.0	2	0.0
3	$sp_3 = \langle e(q_2, p_2) \rangle$	0.0	1	0.0
4	$sp_4 = \langle e(q_2, p_3) \rangle$	0.6	2	1.2

Using Definition 3, we can compute emd^- by adding to *current cost* an estimated lower bound for all remaining feasible paths originating at each vertex q_i, based on the node's remaining capacity (i.e., $cap_{q_i} - f_{q_i}$) and the cost of the last augmented feasible path from q_i. Lemma 2 proves the correctness of the bound. The example of Figure 8 illustrates the computation of emd^-. There are 4 paths already augmented; their costs and iteration order are shown in Table 2. The current cost is 1.2 ($= c_f(sp_4) f(sp_4) = 0.6 \cdot 2$) and the estimated cost of the remaining flows (computed using the nodes of $\mathbf{q}$ in the dashed-line region) is 0.6 ($= c_f(sp_4)(cap_{q_2} - f_{q_2}) = 0.6 \cdot (4-3)$). Thereby, emd^- is 1.8 which is much tighter compared to just using the *current cost* as the lower bound.

Note that the time of updating emd^- during SIA is negligible. Whenever a new feasible path sp from a vertex q_i is augmented (i.e., at each iteration of SIA), we refine emd^- by: (i) adding the augmentation cost of sp ($c_f(sp) \cdot f(sp)$) to the current cost component of emd^-; (ii) subtracting the previous estimated cost for q_i in the estimated cost component; (iii) adding the new estimated cost of q_i. Each of these three increments takes constant time (in fact, (ii) is already cached), so the update of emd^- takes $O(1)$ time.

Summing up, our *progressive bounding* (PB) approach, during the EMD calculation for a candidate, progressively maintains and tightens the running lower bound emd^- and prunes the object as soon as the intermediate emd^- reaches the pruning threshold θ. PB saves unnecessary computations at the latter (expensive) stages of SIA for candidate objects that do not make it to the k-NN result (i.e., conducting only the necessary portion of the entire flow calculation), and thus reduces the cost of each individual refinement.

4.3 Sensitivity to Refinement Order

So far, we have discussed how to terminate a single EMD calculation early by the *progressive bounding* technique. Still, the performance of a k-NN query does not depend only on individual EMD calculations but also on the amount and progresses of EMD calculations. According to Algorithm 1, the EMD for an object $\mathbf{p}$ is essentially refined if all estimated lower bounds $lb_{\mathbf{p}}$ (at the filter phase) are smaller than the pruning threshold θ (line 6 of Algorithm 1), where θ is the k-th best-so-far EMD value.

Figure 9 shows the EMD values of the accessed objects during a 4-NN query execution using Algorithm 1. For each candidate $\mathbf{p}$, a bar shows the real EMD value $emd(\mathbf{q}, \mathbf{p})$, the best lower bound $lb_{\mathbf{p}}$ from the filter phase, and the pruning threshold θ at the time of $\mathbf{p}$'s verification. Every refined candidate can potentially decrease the pruning threshold θ if it replaces another object in the current k-NN set H. For instance, θ is decreased after accessing and refining the 6-th object. Observe that the order by which the objects are accessed is not consistent with their real EMD values. On the other

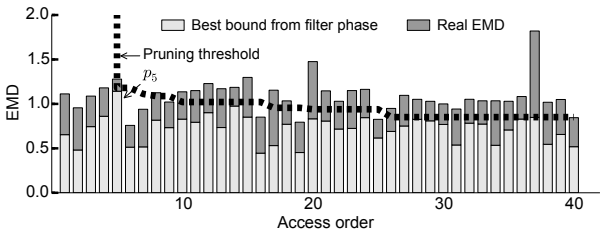


Figure 9: Access order of a k-NN search

hand, the amount of EMD calculations would have been greatly reduced if we had considered the objects in a better order. For instance, if we had accessed the 6-th and 7-th objects before the 5-th object $\mathbf{p}_5$, then $\mathbf{p}_5$ would have been filtered since its best lower bound would have been larger than θ in this case. Note that a better access order not only filters more objects, but also decreases the pruning threshold θ faster such that individual EMD calculations can be *terminated earlier* owing to the *progressing bounding* technique. Unfortunately, the access order of the state-of-the-art filter-and-refinement framework is based on the $getnext$ function (provided by the *normal distribution index* [25] or the *TBI index* [32]), which just returns any unseen object $\mathbf{p}$ having lower bound smaller than a given threshold θ. As shown in this example, there is room to improve the access order such that objects that are likely to have smaller EMD values have higher chances to be refined earlier. Motivated by this analysis, we propose a novel technique that defines and follows a dynamic access order of the candidates.

4.4 Dynamic Refinement Ordering

The main idea behind our *dynamic refinement ordering* (DRO) approach in refining candidates during EMD-based similarity search is to conduct the refinement for multiple candidates *concurrently*. Thus, given a *priority layer* PL set of b candidates ($PL \subseteq \mathcal{D}$), such that each $\mathbf{p} \in PL$ passes all filters based on the current threshold θ, the objective of DRO is to refine all objects in PL concurrently, by augmenting paths to the EMD of $\mathbf{p} \in PL$, which is currently the most *promising* object in PL. Intuitively, an object is promising to augment flows on its corresponding EMD, if the augmentations can make the threshold θ lower or prune objects from PL. For instance, the running lower bound emd^- can be used to prioritize objects, since augmenting an object having the lowest emd^- may update the best k objects found so far and decrease the threshold θ early. The augmentations result in the increase (i.e., tightening) of $emd^-_{\mathbf{p}}$ [6], which may cause another object $\mathbf{p}'$ to take the place of $\mathbf{p}$ as the most promising one and be refined by DRO in the next step. Thus, DRO always refines the object $\mathbf{p}$ with the currently best promising value and checks whether $\mathbf{p}$ can be pruned after updating (i.e., increasing) $emd^-_{\mathbf{p}}$. If this pruning happens, $\mathbf{p}$ is replaced by another candidate object in PL. DRO also keeps track of the *upper bound* $emd^+_{\mathbf{p}}$ for each object $\mathbf{p} \in PL$; if the currently refined object's upper bound becomes smaller than θ, then the current top-k result is updated to include $\mathbf{p}$. The details on how to compute and update an upper bound of a partially computed EMD are given in Section 4.5. DRO continues until PL becomes empty.

Algorithm 3 is a pseudocode of the DRO strategy for EMD-based k-NN search. DRO uses function $getnext\_filter$ to get the next object from $\mathcal{D}$, which passes all filters with respect to the current threshold θ. First, DRO calls this function b times to form the initial PL. PL is stored as a priority queue, in which the top

[6]For the ease of presentation, we denote $emd^-(\mathbf{q}, \mathbf{p})$ by $emd^-_{\mathbf{p}}$, as the query histogram $\mathbf{q}$ is the same for all objects in $\mathcal{D}$.

Algorithm 3 DYNAMIC REFINEMENT ORDERING

```
   H, PL: heap, θ: pruning threshold
   Function getnext_filter(Query q, Index I, Filters Δ)
 1: while I.getnext(q, θ, ⟨p, lb_p⟩) do
 2:     for δ_i ∈ Δ do                                   ▷ Filter phase
 3:         lb_p := max{lb_p, δ_i(q, p)}
 4:         if lb_p ≥ θ then break loop
 5:     if lb_p < θ then return ⟨p, lb_p, ∞⟩

   Algorithm DRO-kNN(Query q, Index I, size b)
 6: θ := ∞; H := ∅; PL := ∅
 7: while |PL| < b do
 8:     PL := PL ∪ getnext_filter(q, I, Δ)
 9: while |PL| ≠ ∅ do
10:     pop ⟨p, emd^-_p, emd^+_p⟩ from PL                 ▷ e.g., lowest emd^-_p
11:     while emd^-_p < PL.top().emd^- do
12:         augment next shortest path in emd(q, p)
13:         if emd^-_p ≥ θ ∨ emd^-_p = emd^+_p then
14:             PL := PL ∪ getnext_filter(q, I, Δ)
15:             break loop
16:         if emd^+_p < θ then
17:             update H to include the new ⟨p, emd^+_p⟩
18:             θ := k-th EMD value in H
19:             for all p ∈ PL such that emd^-_p ≥ θ do
20:                 remove p from PL
21:                 PL := PL ∪ getnext_filter(q, I, Δ)
22:     if loop not broken then
23:         PL := PL ∪ p                                 ▷ add p back to PL
24: return H
```

element is the object with the smallest emd^-. At each iteration, DRO de-heaps the top object $\mathbf{p}$ in PL and progressively refines the current $emd(\mathbf{q}, \mathbf{p})$ by iteratively augmenting shortest paths (i.e., using SIA) while $emd^-_{\mathbf{p}}$ is still smaller than $PL.top().emd^-$ (i.e, the smallest emd^- in PL). A path augmentation increases $emd^-_{\mathbf{p}}$ and decreases $emd^+_{\mathbf{p}}$. If $emd^-_{\mathbf{p}} \geq \theta$, then the current object $\mathbf{p}$ is pruned because in the best case it cannot become better than the current k nearest neighbors (see Section 4.2); DRO calls function $getnext\_filter$ to add another candidate in PL in place of $\mathbf{p}$ (line 14). If $emd^+_{\mathbf{p}} < \theta$, $\mathbf{p}$ is updated in H and the threshold θ is updated accordingly (line 18). After θ is updated, we remove from PL objects whose lower bounds are already greater than or equal to θ (line 19). The removal is facilitated by an additional heap structure that indexes objects in descending order of their lower bounds. If, after some augmentations, $emd^-_{\mathbf{p}}$ becomes no smaller than $PL.top().emd^-$ (line 11) and $\mathbf{p}$ has not been pruned, it is put back to PL (line 23) and the top object of PL takes its place in the inner path augmentation loop. Note that objects that may not be further refined (i.e., condition $emd^-_{\mathbf{p}} = emd^+_{\mathbf{p}}$ at line 13) are either pruned (if $emd_{\mathbf{p}} \geq \theta$) or added to H (if $emd_{\mathbf{p}} < \theta$).

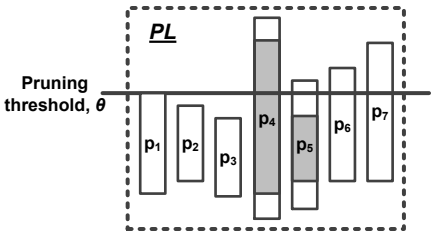


Figure 10: Prioritizing refinement order

Figure 10 illustrates a running instance of DRO. The current lower and upper bounds of each object are indicated by the borders of a bar. Suppose $k = 3$ and there are 7 objects ($= b$) in PL.

Note that the currently best object in PL (in terms of emd^-) is $\mathbf{p}_4$, so DRO refines the current EMD of $\mathbf{p}_4$ by augmenting flows. After the augmentation, the EMD bounds of $\mathbf{p}_4$ are updated to the shaded bar. Note that $emd^-_{\mathbf{p}_5}$ now becomes the lowest emd^-, so the refinement of $\mathbf{p}_4$ is stalled; in the next iteration, $\mathbf{p}_5$ is de-heaped from PL and refined. After augmenting flows to the EMD of $\mathbf{p}_5$, the EMD bounds of $\mathbf{p}_5$ are updated as shown by the shaded bar. Observe that now $emd^+_{\mathbf{p}_5} < \theta$; this causes (i) $\mathbf{p}_5$ to be included in the currently best k objects H and (ii) θ to be updated to $emd^+_{\mathbf{p}_2}$.

DRO is expected to be more efficient than considering the candidates that pass the filters one by one and computing their exact EMD individually. Concurrent refinement by prioritizing candidates with the lowest emd^- performs the first (cheap) iterations of SIA for many candidates and helps in deriving upper bounds emd^+ for them and a good estimate of θ early. Obtaining a tight θ early can help (i) to prune more candidates using the filters (function $getnext\_filter$) and (ii) to avoid the late and expensive path augmentations of SIA for many candidates in PL that can be pruned.

The choice of b (i.e., the size of PL) affects the performance of DRO; if b is very large, DRO performs multiple concurrent SIA executions which may have high memory requirements. If b is too small, the pruning threshold θ does not converge fast to its final value, and more objects enter the refinement phase. Clearly, there is a tradeoff between the performance gain and memory consumption. In our experiments, every SIA thread only constructs a small portion of the entire flow graph. Thereby, b can be set to a relatively large number. For instance, when b is set to 0.2% of the data cardinality, the peak memory consumption of DRO on the largest dataset of our experimental evaluation, WORLD, is just around 20 times of a complete flow graph in a single EMD computation. Moreover, we study a mechanism to avoid worst cases (e.g., every partial graph in PL is full). When the size of PL exceeds a limit, we stop inserting candidates into PL and refine the current EMD by PB until it is pruned or becomes a k-NN candidate. However, in all of our experimental testings, this mechanism is never triggered as the size of PL is much smaller than our default memory limit (i.e., 512MB).

4.5 Running Upper Bound

We now discuss the details of computing and maintaining a *running upper bound* $emd^+_{\mathbf{p}}$ for an object $\mathbf{p}$ whose EMD has partially been computed by SIA. The upper bounds are used by DRO (presented in Section 4.4) to derive a value for θ after having partially computed the EMD of some candidate objects. According to the EMD definition, any flow matrix $\mathbf{F}$ satisfying all three conditions of Equation 2 must lead to an upper bound of the actual EMD. For the ease of discussion, we call a flow matrix $\mathbf{F}$ *possible* if it satisfies all these three optimization constraints; the constraints are satisfied if and only if there is no more feasible path in the flow network. Thus, finding a possible flow matrix $\mathbf{F}$ is equivalent to finding a maximum flow in the network [1], disregarding edge costs.

Based on the above discussion, we can define a running upper bound emd^+ following the same idea of deriving emd^-. Similar to the illustration in Figure 8, emd^+ consists of the *current cost* plus an upper bound considering the nodes which still have remaining flow capacity (i.e., the vertices inside the dashed-line regions). For instance, a maximum flow of the vertices inside the dashed-line region is $(q_1, p_4, 1)$, $(q_2, p_4, 1)$, and $(q_4, p_4, 1)$. The cost of this max flow is 0.7 + 0.9 + 0 = 1.6; thus, emd^+ is 1.2 (*current cost*) + 1.6 = 2.8. Formally, we define the running upper bound as follows.

DEFINITION 4 (RUNNING UPPER BOUND, emd^+). *Consider a flow graph, such that for each node $p_i \in \mathbf{p}$, there is a set of m_i augmented paths $sp_{q_i,1}, \ldots, sp_{q_i,m_i}$. The running upper bound emd^+ is defined as*

$$emd^+ = \sum_{q_i \in \mathbf{q}} \sum_{j=1}^{m_i} c_f(sp_{q_i,j}) \cdot f(sp_{q_i,j}) + \quad \textbf{(current cost)}$$

$$maxflow(\mathbf{q}, \mathbf{p}), \quad \textbf{(maximum flow)}$$

where $maxflow(\mathbf{q}, \mathbf{p})$ returns the cost of a maximum flow based on the remaining capacities of $\mathbf{q}$ and $\mathbf{p}$.

To compute emd^+, we can apply any maximum flow algorithm; however, existing maximum flow algorithms are too expensive [1], considering the fact that emd^+ should be maintained throughout the entire EMD calculation. Instead of using these methods, we propose an efficient greedy approach, which takes advantage of the EMD flow network topology; note that the EMD flow graph is a *complete* bipartite graph between the bins of $\mathbf{q}$ and $\mathbf{p}$. This means that a feasible vertex of $\mathbf{q}$ can always augment flow to any feasible vertex of $\mathbf{p}$ along one edge. Our GreedyUB algorithm (Algorithm 4), for each feasible vertex q_i of $\mathbf{q}$, accesses the feasible vertices p_j of $\mathbf{p}$ in increasing order of $c_{i,j}$ and augments the maximum possible flow along each edge $e(q_i, p_j)$, until the remaining capacity from q_i has been used up. The accumulated cost of the greedily augmented flows is used as the maxflow component $maxflow(\mathbf{q}, \mathbf{p})$ of $emd^+(\mathbf{q}, \mathbf{p})$.

Algorithm 4 GREEDY UPPER BOUND

```
   Algorithm GreedyUB(Histograms q, p)
1: cost := 0
2: while ∃ feasible q_i do
3:     while cap_{q_i} > f_{q_i} do                ▷ q_i has remaining capacity
4:         p_j := feasible bin in p having minimum c_{i,j}
5:         flow := min{cap_{q_i} − f_{q_i}, cap_{p_j} − f_{p_j}}
6:         f_{q_i} := f_{q_i} + flow
7:         f_{p_i} := f_{p_i} + flow
8:         cost := cost + c_{i,j} · flow
9: return cost
```

To facilitate GreedyUB, for each row i of the cost matrix $\mathbf{C}$, we define an order for the bins j based on $c_{i,j}$ (this order is static, i.e., independent of the data and queries, and it is defined once, together with the ground distance function or cost matrix). For example, for row q_1 of the matrix shown in Table 1, the order is $\{p_1, p_3, p_4, p_2\}$. Then, in Algorithm 4 line 4, this order is considered for identifying feasible bins p_j for the current vertex q_i. Thus, the complexity of the greedy algorithm is $O(|E|)$, where E indicates the edges in the graph, since there are at most $|E|$ feasible pairs in the EMD flow graph, i.e., much lower compared to the complexity of standard maximum flow algorithms (e.g., $O(|V||E|)$).

Similar to emd^-, we can update emd^+ incrementally after each path augmentation during the refinement of an EMD. After every augmentation of SIA, according to the capacity feasibility, we cancel and re-augment (for those that become feasible after the cancelation) the corresponding flows in the maximum flow graph formed by the greedy algorithm. The number of canceled/re-augmented edges in the maximum flow instance is at most equal to the flow units f just being augmented in the current EMD flow. Thereby, the maintenance cost of emd^+ after each augmentation is $O(f)$.

5. EXPERIMENTAL EVALUATION

In this section, we conduct extensive experiments to evaluate the performance of our proposed EMD-based similarity search framework and compare it with state-of-the-art solutions. All methods are implemented in C++ and evaluated on 3.40 GHz quad-core machines running Ubuntu 12.04, with 16 GBytes of main memory.

Table 3: The statistics of six real datasets

Name	# Objs	# Bins	Description
RETINA	3932	96 (12×8)	Feline retina images. Default dataset in [25, 30, 32].
IRMA	10K	199	Medical images from IRMA[7] project. Used in [25, 30, 32].
FLICKR	680K	100 (10×10)	Images crawled from Flickr[8]. This dataset is used in [25].
PANORAMIO	500K	576 (24×24)	Images of European cities from Panoramio[9].
FRIENDS	320K	768 (24×32)	Images captured every 25 frames from the TV series "Friends".
WORLD	3M	1024 (32×32)	Images from ImageNet[10] project.

We use six real datasets in our evaluation; their default statistics are shown in Table 3. The RETINA, IRMA and FLICKR histogram sets are taken from [25]. The PANORAMIO, FRIENDS and WORLD histogram sets are generated by the same method with RETINA and FLICKR as suggested in [25]: each image is divided into tiles by a grid (e.g., 24×24 square granularity); for each tile, the 12-feature MPEG-7 color layout descriptor (CLD) is extracted, and we pick only the first feature as the value of the histogram bin. Similar to [25,30,32], we use Euclidean distance as the ground distance (cost matrix) in all the experiments. Note that our techniques are not restricted to any specific ground distance.

In the evaluation, we compare a set of EMD computation methods when used as a black-box module of the filter-and-refinement framework (see Section 2.2). In all cases, the filtering of candidates is done by applying the state-of-the-art filtering techniques as used in [25]. First, we use the normal distribution index [25] to implement the $I.getnext$ function. For each retrieved candidate by this function, we apply the following set of filters Δ in this order: full projection [7], reduced dimension [30] (only for RETINA and IRMA, on which the filter has positive effects as in [25]), and independent minimization [5]. For the candidates that pass the filter phase, we compare the application of the following EMD computation methods: (i) capacity scaling (CAP), (ii) cost scaling (COS), (iii) transportation simplex (TRA), (iv) network simplex (NET), (v) SSP (Section 2.1), (vi) SIA (Section 3). In addition, we evaluate our progressive bounding (PB) and dynamic refinement ordering (DRO) techniques when applied in conjunction with SIA. At each experimental instance (e.g., for a given dataset and k), we run 100 k-NN queries choosing $\mathbf{q}$ randomly from the corresponding dataset and average the query cost. The 100 queries for RETINA, IRMA and FLICKR dataset are the same queries used in [25, 30, 32]. The default value of k is 32 and the default b (i.e., maximum size of PL in DRO) is set to 0.2% of the corresponding dataset cardinality.

5.1 Performance Improvement

First, we compare the performance of different black-box EMD computation methods in the filter-and-refinement framework. Table 4 shows the average query time of the six EMD computation methods for 32-NN similarity queries on the six datasets. SIA is the best method which constantly outperforms its base method SSP and other alternatives (e.g., TRA) by at least 2.4 times. In addition, SIA scales well with the problem dimensionality (i.e., the number of histogram bins and the cardinality of datasets). Figure 11 shows

[7]http://ganymed.imib.rwth-aachen.de/irma
[8]http://www.flickr.com
[9]http://www.panoramio.com
[10]http://www.image-net.org

Table 4: Comparison of black-box EMD computation methods

		RETINA	IRMA	FLICKR	PANO.	FRIENDS	WORLD
Query time	CAP	0.68s	3.90s	7.89s	803s	3279s	22715s
	COS	1.13s	6.48s	13.58s	1394s	5628s	48917s
	NET	0.52s	4.13s	7.04s	763s	3573s	22774s
	SSP	0.57s	6.14s	7.52s	2048s	10370s	65731s
	TRA	0.57s	8.30s	7.72s	3395s	15307s	137414s
	SIA	0.17s	1.51s	2.77s	318s	1362s	7082s
Filter time		0.006s	0.008s	0.749s	1.009s	1.476s	12.238s

the performance of the methods as a function of k on five datasets: RETINA, IRMA, PANORAMIO, FRIENDS, and WORLD.[11] SIA outperforms SSP by around an order of magnitude, while being at least two times faster than the runner-up network simplex (NET). Note that the relative performance difference between methods is not very sensitive to k. SIA is obviously the best module for calculating EMD under the filter-and-refinement framework. Note also that the time spent in the filter phase (in Table 4) is negligible, which confirms our discussion that the *refinement phase dominates the runtime* of the entire filter-and-refinement framework.

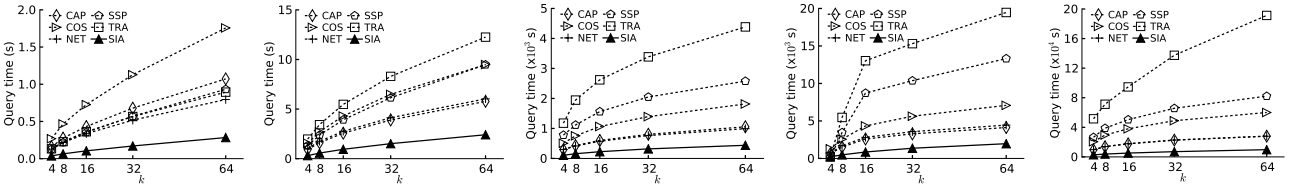
Next, we evaluate the improvement offered by our progressive bounding and dynamic refinement ordering techniques. Figure 12 compares the runtime of k-NN queries when (i) using SIA as a black-box module, (ii) applying progressive bounding in SIA (PB), and (iii) applying dynamic refinement ordering in conjunction with progressive bounding in SIA (DRO). Observe that PB achieves a large performance improvement over SIA (i.e., the best black-box module in Figure 11). The improvement generally becomes more visible when the problem size increases (i.e., the number of bins and the cardinality of datasets). This can be explained by the fact that the size of the flow graph is quadratic to the number of histogram bins, therefore the latter iterations of SIA become extremely expensive. The progressive bounding technique helps to avoid reaching these iterations, as explained in Section 4.2. DRO offers a stable improvement over PB (40%–60%), indicating that the concurrent refinement and dynamic reordering techniques have positive impact on the performance.

In summary, DRO is the recommended methodology for EMD-based similarity queries, being several times to two orders of magnitude faster than using off-the-shelf EMD computation methods. Notably, *all previous works* [5,25,30,32] adopt *transportation simplex* (TRA) for calculating EMD at the refinement phase. As shown in our evaluation, this method only performs well when the number of bins is relatively small due to its exponential time complexity.

Room for Improvement. So far, we have shown the superiority of our PB and DRO techniques over existing approaches. To see how much room for improvement exists for EMD-based similarity queries, we compare PB and DRO to two ideal yet practically infeasible methods, ESS and OI. Assuming that we know the result of a k-NN query by an oracle, ESS (Essential k-NN search) calculates the EMDs of only the exact k-NNs, representing the lowest possible effort to compute the results of a k-NN query. OI (Oracle Index) assumes that an *oracle* index is available which offers an optimal $getnext$ function that returns the objects in the ascending order of their *exact* EMDs. OI can be viewed as an optimal ordering method integrated with the current state-of-the-art filtering techniques. In our experiments, both ESS and OI calculate EMDs by the best black-box module (SIA), and OI uses our progressive bounding (PB) technique to terminate EMD calculations early.

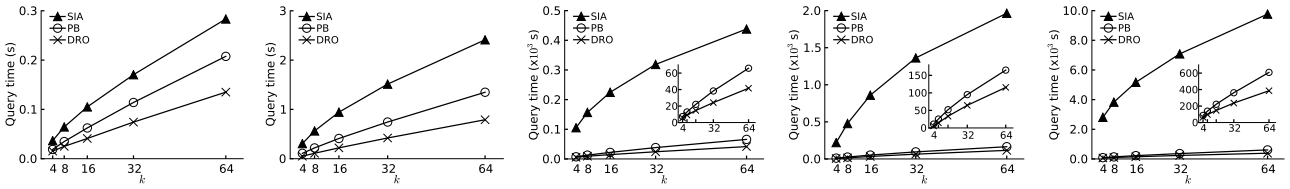
Figure 13 shows the query time of four methods, PB, DRO, ESS, and OI, as a function of k. First, we observe that DRO's perfor-

[11]The results on FLICKR are similar to those of RETINA.



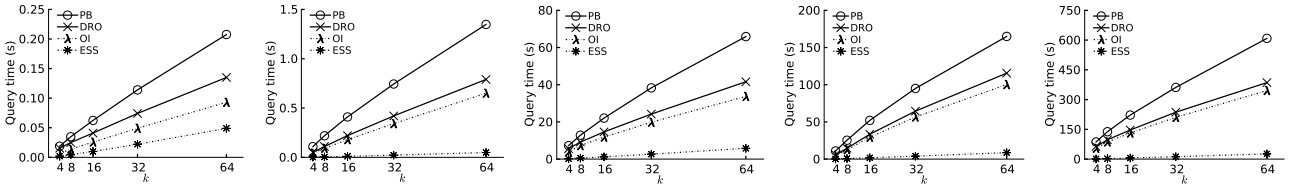
(a) RETINA (b) IRMA (c) PANORAMIO (d) FRIENDS (e) WORLD

Figure 11: Performance of black-box EMD computation methods varying on k



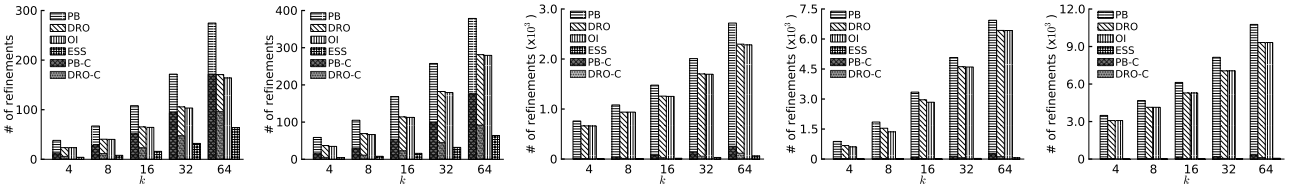
(a) RETINA (b) IRMA (c) PANORAMIO (d) FRIENDS (e) WORLD

Figure 12: Performance of SIA, PB and DRO varying on k



(a) RETINA (b) IRMA (c) PANORAMIO (d) FRIENDS (e) WORLD

Figure 13: Closeness of PB and DRO to the oracle methods (query time)



(a) RETINA (b) IRMA (c) PANORAMIO (d) FRIENDS (e) WORLD

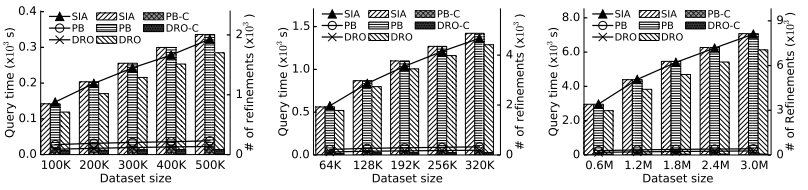
Figure 14: Closeness of PB and DRO to the oracle methods (number of refinements)

mance is very close to that of OI, improving PB by a significant extent towards an ideal method. Figure 14 illustrates the number of EMD refinements[12] of these four methods, and also the number of *complete* refinements (i.e., the EMD calculations that are fully conducted) of PB and DRO (denoted by PB-C and DRO-C respectively), varying on k. These experiments confirm the robustness of our dynamic reordering technique as DRO commences almost the same number of EMDs compared to the optimal method OI. Besides, in Figure 14, PB-C and DRO-C are very close to ESS (i.e., k). In specific, DRO-C is only slightly larger than k, indicating that only very few objects that are not the actual k-NN results need to be fully refined by DRO, which again verifies the effectiveness of DRO. Obviously, there is limited room for improving DRO, using the current state-of-the-art filtering techniques, as its performance is already close to that of OI. On the other hand, we believe that there is still room for improving upon the current filtering methods as the performance gap between OI and ESS is still significant.

[12]An object is counted even when only one path is augmented.

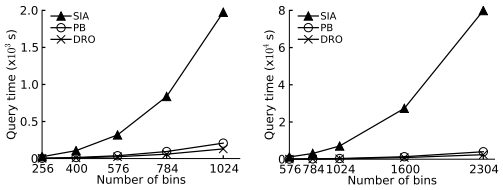
5.2 Scalability Experiments

In the first scalability study, we conduct experiments on PANORAMIO subsets with cardinality 100K to 500K, on FRIENDS subsets (64K to 320K), and on WORLD subsets (0.6M to 3M), by randomly picking objects from the corresponding datasets. The lines in Figure 15 show the query time of three methods, SIA, PB, and DRO, as a function of the object cardinality, after setting $k = 32$. PB and DRO scale very well with the database size (they are almost insensitive), since the progressive bounding technique benefits from better pruning thresholds owing to the increasing number of objects. This experiment exposes an important advantage of our approach: its performance is less sensitive to the database size, while k is the main cost factor. Note that while the database size may increase arbitrarily, in practice k is not expected to grow at the same rate, since similarity search queries typically retrieve a limited number of objects. The bars in Figure 15 show the number of refinements of SIA, PB, and DRO (note that in PB and DRO, only a portion of these are *complete refinements*, denoted



(a) PANORAMIO (b) FRIENDS (c) WORLD

Figure 15: Scalability to dataset cardinality (number of objects)



(a) PANORAMIO (b) WORLD

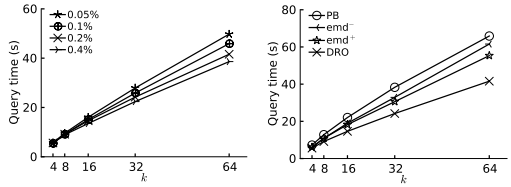
Figure 16: Scalability to histogram dimensionality

by PB-C and DRO-C). We observe that SIA is quite sensitive to the number of refinements while PB and DRO are likely sensitive to the number of the complete refinements (PB-C and DRO-C). In addition, the improvement of DRO over PB is stable, indicating that the reordering technique is not affected by the increasing number of objects that survive the filters.

In the second scalability study, we evaluate the effectiveness of our techniques against the histogram dimensionality (i.e., number of bins). We generate histogram sets of different dimensionalities (square grid granularities) up to 2304, on PANORAMIO and WORLD. Figure 16 shows the query time as a function of the histogram dimensionality on the two datasets ($k = 32$). We observe that the query time of PB and DRO increases more slowly compared to that of SIA. The graceful scalability of our methods w.r.t. histogram dimensionality enables the application of EMD-based similarity search on high-dimensional histogram representations (e.g., multi-dimensional features with fine partitioning).

5.3 Parameter Tuning in DRO

Finally, we demonstrate the effect of two sensitive parameters in DRO, i.e., the maximum number of concurrently refined objects b (the size of PL), and the function to prioritize objects in DRO.



(a) Size of PL, b (b) Prioritizing function

Figure 17: Performance tuning of DRO on PANORAMIO

Figure 17(a) shows the average query time of k-NN queries on the PANORAMIO dataset, for different values of b (as different series). DRO becomes faster when b increases, which indicates that concurrent refinement is an effective strategy to evaluate EMD-based k-NN queries. On the other hand, as discussed in Section 4.4, the memory requirements of DRO are proportional to b, so the performance improvement comes with a memory tradeoff.

As mentioned in Section 4.4, the running lower bound emd^- can be used to prioritize objects in DRO. However, as shown in Figure 17(b), emd^- is not an effective prioritizing function since it does not take the bound tightness into consideration. For instance, assume that both $emd^-_{\mathbf{p_a}}$ and $emd^-_{\mathbf{p_b}}$ have the same value. Intuitively, if $emd^+_{\mathbf{p_a}} < emd^+_{\mathbf{p_b}}$, $\mathbf{p_a}$ should be given higher priority than $\mathbf{p_b}$ during the refinement, because $\mathbf{p_a}$ has a tighter bound and it is more likely to be added to the current k-NN set H and decrease θ. Thus, DRO (in Figure 17(b) and the previous experiments) uses the following prioritizing function, which is a slight modification of emd^- (Definition 3): $(1-\alpha)\cdot$*current cost* + $\alpha\cdot$*estimated cost*, where α indicates the tightness of the running bounds (i.e., $\frac{emd^+ - emd^-}{emd^+}$). As Figure 17(b) shows, this function, when used in DRO, maintains a constant advantage over PB, compared to prioritizing by just using emd^- or emd^+.

6. RELATED WORK

Earth Mover's Distance (EMD) was first introduced by the computer vision community as an effective similarity metric [23]. EMD is also known as Mallows distance or Wasserstein distance in statistics [15]. As a cross-bin distance, EMD matches better the human perception of differences [22], compared to bin-by-bin distances like Euclidean distance or χ^2-statistic. EMD supports analysis and search in a wide range of application domains, such as image retrieval [23,24], computer vision [11,17,21], machine learning [6,9], probabilistic [25, 32] and multimedia databases [5, 30], video and music identification [28, 31], phishing detection [10], data cleaning [8], privacy [16], matrix factorization [26], clustering [4, 6], classification [14], etc. An empirical study [20] that compares nine families of image dissimilarity measures based on distributions of color and texture features shows that EMD has the best overall quality among the others, yet also the highest computational cost.

Due to the usefulness of EMD, the database community developed techniques for similarity queries based on this measure. Work in this direction is based on the filter-and-refinement framework [5, 25, 30, 32], described in Section 2.2. The main focus is the development of fast and effective lower bounds for EMD, which help in pruning objects that cannot make it in the result, at the filter phase. Assent et al. [5] were the first to study EMD in the filter-and-refinement framework. Wichterich et al. [30] propose a dimensionality reduction technique, showing that the EMD of two objects in the derived space is a lower bound for their actual EMD. Xu et al. [32] propose a lower bound of EMD based on primal-dual techniques [1] from linear programming and use B$^+$-tree indexing to support the filter phase of k-NN and range queries. Ruttenberg and Singh [25] develop a new index structure for EMD-based similarity queries based on the projection lower bound in [7]. All these solutions focus only on the effectiveness of filters and efficiency of the filter phase, simply treating the refinement phase as a black-box process. However, as we have shown in this paper, the refinement cost *dominates* the overall cost and deserves more attention.

Besides these results by the database community, there are also studies on efficient EMD approximations [2, 3, 13, 19, 27]. Andoni et al. [3] study EMD approximations in high-dimensional spaces. Pele and Werman [19] accelerates the EMD computation by limiting the number of edges in the flow graph, without guarantees of the quality of approximation. Both [13] and [27] study linear-time approximations of EMD, and [2] proposes sketches for approximating planar EMD. However, these works assume either specific ground distances (e.g., L_1 norm) [2, 3, 27] and bin spaces (e.g., $\mathbb{R}^2$) [2, 3, 13, 27], or certain histogram types (e.g., dominant color descriptor) [13], which do not apply to the general setup of EMD.

For example, the choice of ground distance could be application-dependent [19, 22, 24]. Note that our proposed techniques do not have any of the above restrictions. Moreover, there is no study on filtering bounds and indexing structures on top of these approximation techniques. Therefore, using the approximation methods, a similarity query may have to compute EMDs between the query and all the objects in the database, which is not scalable for large datasets. Finally, the acceleration of EMD under Manhattan network when using L_1 as the ground distance is studied in [18].

7. CONCLUSION

In this paper, we studied the efficient evaluation of similarity queries using Earth Mover's Distance (EMD). First, we showed how we can adapt SIA, an algorithm originally proposed for spatial matching problems, to compute the EMD between two histograms efficiently. Then, we proposed a progressive refinement strategy, which updates a lower bound for EMD during its computation, in order to abandon early a partial EMD refinement, if the object cannot make it in the query result. Finally, we proposed a technique which concurrently handles the refinement of multiple candidates, by dynamically reordering them and computing upper bounds that help to tighten the pruning threshold early. Our experiments show that our methods are very effective in practice, decreasing the overall cost of EMD-based similarity queries by up to two orders of magnitude, compared to the state-of-the-art solution [25].

Note that although our discussion assumes that the histograms are normalized to sum up to the same values, EMD can also be applied in cases where histograms are not normalized (in this case, a slightly different definition than that of Equation 2 is used to reflect the maximum flow that could be sent from $\mathbf{q}$ to $\mathbf{p}$). In addition, it is not necessary that the two histograms have the same number or locations of bins. Our framework and solutions are not sensitive to these problem variants. Finally, note that our solution can also be used for evaluating *range* similarity queries, where the objective is to retrieve objects whose EMD to $\mathbf{q}$ does not exceed a given threshold ϵ. In this case, SIA and PB can directly be applied, however, DRO is not relevant because the threshold is fixed and insensitive to the execution order.

In the future, we plan to study the optimization of the refinement step in similarity queries based on other expensive distance measures (such as dynamic time warping).

8. ACKNOWLEDGMENT

This work is supported by grants HKU 714212E, 711110, and 711309E from Hong Kong RGC. Leong Hou U is supported by grant MYRG109(Y1-L3)-FST12-ULH. We would like to thank Brian Ruttenberg for providing part of his code on [25] and the anonymous reviewers for their insightful comments.

9. REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms and applications*. Prentice Hall, 1993.

[2] A. Andoni, K. D. Ba, P. Indyk, and D. P. Woodruff. Efficient sketches for earth-mover distance, with applications. In *FOCS*, pages 324–330, 2009.

[3] A. Andoni, P. Indyk, and R. Krauthgamer. Earth mover distance over high-dimensional spaces. In *SODA*, pages 343–352, 2008.

[4] D. Applegate, T. Dasu, S. Krishnan, and S. Urbanek. Unsupervised clustering of multidimensional distributions using earth mover distance. In *KDD*, pages 636–644, 2011.

[5] I. Assent, A. Wenning, and T. Seidl. Approximation techniques for indexing the earth mover's distance in multimedia databases. In *ICDE*, page 11, 2006.

[6] M. H. Coen, M. H. Ansari, and N. Fillmore. Comparing clusterings in space. In *ICML*, pages 231–238, 2010.

[7] S. Cohen and L. Guibas. The earth mover"s distance: Lower bounds and invariance under translation. Technical report, Stanford University, 1997.

[8] T. Dasu and J. M. Loh. Statistical distortion: Consequences of data cleaning. *PVLDB*, 5(11):1674–1683, 2012.

[9] N. Ferns, P. S. Castro, D. Precup, and P. Panangaden. Methods for computing state similarity in markov decision processes. In *UAI*, pages 174–181, 2006.

[10] A. Y. Fu, L. Wenyin, and X. Deng. Detecting phishing web pages with visual similarity assessment based on earth mover's distance (EMD). *IEEE Trans. Dependable Sec. Comput.*, 3(4):301–311, 2006.

[11] K. Grauman and T. Darrell. Fast contour matching using approximate earth mover's distance. In *CVPR (1)*, pages 220–227, 2004.

[12] F. S. Hillier and G. J. Lieberman. *Introduction to Mathematical Programming*. McGraw-Hill, 1990.

[13] M.-H. Jang, S.-W. Kim, C. Faloutsos, and S. Park. A linear-time approximation of the earth mover's distance. In *CIKM*, pages 505–514, 2011.

[14] H. J. Karloff, S. Khot, A. Mehta, and Y. Rabani. On earthmover distance, metric labeling, and 0-extension. In *STOC*, pages 547–556, 2006.

[15] E. Levina and P. J. Bickel. The earth mover's distance is the mallows distance: Some insights from statistics. In *ICCV*, pages 251–256, 2001.

[16] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, pages 106–115, 2007.

[17] P. Li, Q. Wang, and L. Zhang. A novel earth mover's distance methodology for image matching with gaussian mixture models. In *ICCV*, 2013 (to appear).

[18] H. Ling and K. Okada. An efficient earth mover's distance algorithm for robust histogram comparison. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(5):840–853, 2007.

[19] O. Pele and M. Werman. Fast and robust earth mover's distances. In *ICCV*, pages 460–467, 2009.

[20] J. Puzicha, Y. Rubner, C. Tomasi, and J. M. Buhmann. Empirical evaluation of dissimilarity measures for color and texture. In *ICCV*, pages 1165–1172, 1999.

[21] Z. Ren, J. Yuan, and Z. Zhang. Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera. In *ACM Multimedia*, pages 1093–1096, 2011.

[22] Y. Rubner and C. Tomasi. *Perceptual Metrics for Image Database Navigation*. Kluwer Academic Publishers, 2001.

[23] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *ICCV*, pages 59–66, 1998.

[24] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[25] B. E. Ruttenberg and A. K. Singh. Indexing the earth mover's distance using normal distributions. *PVLDB*, 5(3):205–216, 2011.

[26] R. Sandler and M. Lindenbaum. Nonnegative matrix factorization with earth mover's distance metric. In *CVPR*, pages 1873–1880, 2009.

[27] S. Shirdhonkar and D. W. Jacobs. Approximate earth mover's distance in linear time. In *CVPR*, pages 1–8, 2008.

[28] R. Typke, P. Giannopoulos, R. C. Veltkamp, F. Wiering, and R. van Oostrum. Using transportation distances for measuring melodic similarity. In *ISMIR*, pages 107–114, 2003.

[29] L. H. U, K. Mouratidis, M. L. Yiu, and N. Mamoulis. Optimal matching between spatial datasets under capacity constraints. *ACM Trans. Database Syst.*, 35(2), 2010.

[30] M. Wichterich, I. Assent, P. Kranen, and T. Seidl. Efficient EMD-based similarity search in multimedia databases via flexible dimensionality reduction. In *SIGMOD*, pages 199–212, 2008.

[31] J. Xu, Q. Bai, Y. Gu, A. K. H. Tung, G. Wang, G. Yu, and Z. Zhang. EUDEMON: A system for online video frame copy detection by earth mover's distance. In *ICDE*, pages 1233–1236, 2012.

[32] J. Xu, Z. Zhang, A. K. H. Tung, and G. Yu. Efficient and effective similarity search over probabilistic data based on earth mover's distance. *PVLDB*, 3(1):758–769, 2010.